Implementación y Validación del Algoritmo de Robótica de Enjambre *Particle Swarm Optimization* en Sistemas Físicos

Alex Daniel Maas Esquivel

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Implementación y Validación del Algoritmo de Robótica de Enjambre *Particle Swarm Optimization* en Sistemas Físicos

Trabajo de graduación presentado por Alex Daniel Maas Esquivel para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2021

Vo.Bo.:

(f) _____

Ing. Luis Rivera

Tribunal Examinador:

(f) _____

Ing. Luis Rivera

(f) _____

(f) _____

Fecha de aprobación: Guatemala, de de .

Prefacio

La elaboración de la presente tesis surge del interés personal de poder poner en practica los conocimientos adquiridos durante estos últimos años acerca de programación, inteligencia de enjambre, robótica, sistemas de control y poder aplicarlos en temas de actualidad como lo es la robótica *swarm* o robótica de enjambre, un tema del cual aun se puede profundizar mucho más y cuenta con una cantidad ilimitada de aplicaciones.

Quiero agradecer principalmente a Dios y a mis padres por haberme apoyado desde el momento que decidí estudiar en la capital y por nunca abandonarme durante todo este proceso, a mis hermanos por su apoyo. Agradezco a la Universidad del Valle de Guatemala y a sus catedráticos que me han brindado su conocimiento, principalmente me gustaría agradecer a mi asesor de tesis el Dr. Luis Alberto Rivera por el tiempo dedicado a resolver mis dudas y compartir su conocimiento para la realización de esta tesis.

Finalmente me gustaría agradecer a todos los amigos que tuve la oportunidad de hacer durante estos 5 años, amistades que se mantendrán para toda la vida, gracias por las risas, los momentos, los viajes compartidos, etc. sin duda hicieron mi experiencia en la Universidad mucho mejor.

Índice

Lista de figuras

Lista de cuadros

Resumen

La robótica de enjambre emplea patrones de comportamiento colectivo mediante iteraciones entre robots y robots con su entorno de forma de alcanzar objetivos o metas establecidas, el algoritmo *Particle Swarm Optimization* (PSO) es utilizado como planificador de trayectorias y por medio de una variedad de controladores se busca una trayectoria suave y controlada hacia el punto de meta.

Se hace uso del *Modified Particle Swarm Optimization* (MPSO) desarrollado en la fase II [1] así como los hiperparametros obtenidos de la aplicación de redes neuronales en la fase III [2], ambos trabajos se realizaron a nivel de simulaciones por lo cual se procede a realizar su implementación en un sistema físico. Se opta por usar el ordenador *Raspberry* para su implementación debido a su afinidad con robots móviles como el Pi-puck y posteriormente una plataforma móvil propia de la UVG haciendo uso de la *Raspberry*.

Debido a que la plataforma móvil aún se encuentra en desarrollo, se desarrolla un sistema de comunicación haciendo uso de una red local y por medio de un módulo GPS se obtienen las coordenadas de cada agente en tiempo real. Para ambas estrategias se hace uso de programación multihilos para que se puedan ejecutar ambos procesos al mismo tiempo que el PSO. Para realizar una correcta validación de este proceso se realizan pruebas en la mesa de pruebas UVG.

Abstract

Swarm robotics employs collective behavior patterns through iterations between robots and robots with their environment in order to achieve established objectives or goals, the Particle Swarm Optimization (PSO) algorithm is used as a trajectory planner and through a variety of drivers are looking for a smooth and controlled trajectory to the goal point.

The Modified Particle Swarm Optimization (MPSO) developed in phase II [1] is used as well as the hyperparameters obtained from the application of neural networks in phase III [2]. its implementation in a physical system. It was decided to use the Raspberry computer for its implementation due to its affinity with mobile robots such as the Pi-puck and later a mobile platform owned by the UVG using the Raspberry.

Because the mobile platform is still under development, a communication system is developed using a local network and through a GPS module the coordinates of each agent are obtained in real time. For both strategies, multithreaded programming is used so that both processes can be executed at the same time as the PSO. To carry out a correct validation of this process, tests are carried out on the UVG test table.

CAPÍTULO 1

Introducción

El algoritmo de robótica de enjambre MPSO así como la obtención y optimización de hiperparámetros haciendo uso de redes neuronales funcionan bastante bien a nivel de simulación. En este trabajo se presenta una implementación en un sistema físico así como pruebas de validación realizadas en la mesa de pruebas UVG. Se busca replicar los resultados obtenidos a nivel de simulación de fases anteriores mientras el algoritmo es ejecutado en el ordenador *Raspberry*, el cual es el elegido para esta implementación.

A diferencia del proceso en simulación no se cuenta aún con una plataforma móvil la cual usar para la implementación (esta se encuentra en desarrollo) por lo que se implementa programación multihilos para la ejecución de un proceso de comunican entre agentes haciendo uso de una red local y la obtención de coordenadas por medio de un módulo GPS. Se toman como base las dimensiones del robot móvil E-puck para realizar la implementación de los diversos controladores, los cuales se encargan de mantener una trayectoria suave y controlada. Estas dimensiones se deberán modificar al momento de lista la plataforma móvil, así como el tipo de controlador y función *fitness* o función costo se pueden modificar de igual forma en la sección de declaración de variables.

CAPÍTULO 2

Antecedentes

2.1. Robotarium de Georgia Tech

El proyecto de Robotarium proporciona una plataforma de investigación robótica de enjambre de acceso remoto. En esta se permite a personas de todo el mundo hacer diversas pruebas con sus algoritmos de robótica de enjambre a sus propios robots con fines de apoyar la investigación y seguir buscando formas de mejorar las trayectorias definidas. Para utilizar la plataforma se debe descargar el simulador que se encuentra en la pagina del Robotarium el cual presenta la opción de descargarlo en Matlab o Python, se debe registrar en la página del Robotarium y esperar a ser aprobado para crear el experimento [3].

Se presenta una base plana con bordes que limitan el movimiento de los robots a utilizar, estas cumplen la función de fronteras del espacio de búsqueda. La base es de un color blanco para facilitar el procesamiento de imágenes y se cuenta con una cámara ubicada en el techo apuntando a toda la base, con esta lograr capturar las posiciones de los diferentes robots de pruebas [3].

Figura 1: Mesa de Pruebas del Robotarium de Georgia Tech [3].

En el departamento de Ingeniería Electrónica, Mecatrónica y Biomédica de la Universidad del Valle de Guatemala se tuvo la iniciativa de investigar, aprender y trabajar con la inteligencia de enjambre (*Robótica Swarm*), gracias a esto surgieron las fases 1, 2 y 3 del conocido "Megaproyecto Robotat".

2.2. Megaproyecto Robotat

En la fase I se procedió con el diseño y la elaboración de una plataforma para el Robotat donde se implementó un algoritmo de visión de computadora para poder obtener la posición y orientación del robot desde una perspectiva planar (2 dimensiones). Posteriormente en la fase II desarrollada por Aldo Aguilar se tomo el modelo estándar y existen del PSO y se procedió a modificarlo para que este tome en consideración las dimensiones de robots físicos y la velocidad a la que estos pueden moverse, además se implementaron diferentes tipos de controladores para buscar el robot pueda llegar al punto de meta realizando una trayectoria suave y controlada.

Se realizaron distintas pruebas buscando encontrar el mejor conjunto de hiperparámetros y probando distintas funciones objetivo, todo esto para hacer un algoritmo mucho más completo a nivel de simulación haciendo uso del programa Webots.

2.3. PSO

En la tercera fase, desarrollada por Eduardo Santizo se mejoró el desempeño del algoritmo de optimización de partículas PSO haciendo uso del método denominado *PSO Tunner*, el

cual consiste de una red neuronal recurrente la cual es capaz de tomar diferentes métricas de las partículas PSO y tornarlas atravesé de su procesamiento por medio de una red LSTM, GRU o BiLSTM y busca realizar una predicción de los hiper parámetros que debería emplear el algoritmo. Esta predicción es de carácter dinámico, lo que hace que en cada iteración se generan las métricas que describen al enjambre y use estos resultados para la siguen te iteración buscando así reducir el tiempo de convergencia y susceptibilidad a mínimos locales del PSO original [2].

En la misma fase se elaboró una alternativa al algoritmo de navegación alrededor de un ambiente conocido por medio de programación dinámica basándose en el ejemplo de programación dinámica Gridworld. Este nos presenta el caso donde un agente se mueve dentro de un espacio de estados representado en la forma de una cuadrícula y únicamente puede realizar 4 movimientos: Moverse hacia arriba, abajo, izquierda o derecha. De acuerdo a su estado actual y último movimiento es capaz de moverse a un nuevo estado y obtiene una recompensa, el agente busca obtener el máximo de recompensas posibles generando así una ruta óptima desde cada estado hasta la meta [2].

Finalmente, estos avances fueron compactados en un grupo de herramientas llamado Swarm Robotics Toolbox para ser usado en futuras fases. Debido a la pandemia ocasionada por el COVID-19, esta nueva propuesta de algoritmo únicamente fue probada a nivel de simulación, haciendo uso del programa Webots para las diversas pruebas realizadas con el robot e-puck.

2.4. Ant Colony

En la tercera tesis, desarrollada por Gabriela Iriarte se buscó crear una alternativa al Modified Particle Swarm Optimization (MPSO) para su futuro uso en los Bitbots de la UVG, desarrollando así el algoritmo de planificador de trayectorias Ant Colony. Se implemento el algoritmo Simple Ant Colony, y después el algoritmo Ant System [4].

Se emplearon diversos controladores y se modificó el camino encontrado interpolándolo para crear más metas y lograr una trayectoria suave. Los parámetros encontrados se validaron por medio de simulaciones computarizadas permitiendo visualizar el comportamiento de colonia y adaptar los modelos del movimiento y de la cinemática a los Bitbots [4]. Lamentablemente por la pandemia ocasiona por el COVID-19 igual que el caso del PSO esta nueva propuesta de algoritmo se vio limitada a pruebas únicamente en simulación por medio del programa Webots.

CAPÍTULO 3

Justificación

La parte primordial de cada nueva propuesta de algoritmo es su validación en una aplicación y ambiente real. Como proyecto la Universidad del Valle de Guatemala y su departamento de Ingeniería Macatrónica, Electrónica y Biomédica han buscado introducirse al mundo de la robótica Swarm y poder desarrollar su propio laboratorio enfocado tanto para la enseñanza como para la investigación.

El enfoque principal de esta tesis, es dar un paso más hacia la creación de este laboratorio, tomando conocimientos ya existentes de fases anteriores y poder darles un cierre adecuado cumpliendo todos los objetivos previos propuestos para los algoritmos de robótica Swarm pero aplicados en un ambiente real.

Para esta tesis se busca poder realizar la correcta migración del algoritmo de optimización de partículas PSO, que actualmente se encuentra en fase de simulación (Webots), y por medio de diferentes pruebas poder verificar el correcto funcionamiento de este y validar la correcta migración del algoritmo hacia una plataforma y entorno real, donde será posible su futuro uso en robots físicos comerciales o desarrollados por el mismo departamento y poder usarlos en la mesa de pruebas ya existente en la Universidad del Valle de Guatemala.

CAPÍTULO 4

Objetivos

Objetivo General

Implementar y validar el algoritmo de robótica de enjambre *Particle Swarm Optimization* desarrollado en años anteriores, a nivel de simulación, en sistemas físicos.

Objetivos Específicos

- Evaluar distintas opciones de microcontroladores, sistemas embebidos, lenguajes de programación, entornos de desarrollo y robots móviles, para seleccionar los más adecuados para su uso en aplicaciones de robótica de enjambre, utilizando específicamente el PSO.
- Migrar el algoritmo de robótica de enjambre PSO hacia el microcontrolador del sistema físico seleccionado.
- Validar la migración del algoritmo de robótica de enjambre y verificar el desempeño de los sistemas físicos mediante pruebas simples en ambientes controlados.

CAPÍTULO 5

Alcance

El alcance de este trabajo de graduación consiste en la implementación y validación del algoritmo de robótica de enjambre *Particle Swarm Optimization* en sistemas físicos, para lo cual se toma el modelo PSO modificado [1] así como los hiperparametros resultantes del uso de redes neuronales [2] para la implementación de este algoritmo en un sistema físico. Se opta por realizar la implementación de este algoritmo haciendo uso del ordenador *Raspberry Pi* que entre sus muchas ventajas destaca su afinidad hacia los robots móviles.

Como consideraciones para el uso del algoritmo PSO se tiene la recepción y envió de información a otros agentes, la capacidad de obtener la posición actual de cada agente en tiempo real, conocer las dimensiones del robot móvil y contar con una cantidad N de agentes para poder validar el correcto funcionamiento del algoritmo.

Por limitaciones de la pandemia COVID-19 y la falta de una plataforma móvil integrada con módulos de comunicación, llantas y sistema GPS, se plantean diversas técnicas adicionales para cumplir con estos requerimientos para demostrar el funcionamiento del algoritmo PSO. En futuras fases del proyecto se pueden implementar el código desarrollado, modulo GPS y sistema de comunicación en una plataforma móvil para observar su respuesta en la mesa de pruebas UVG.

CAPÍTULO 6

Marco teórico

6.1. Robótica Swarm

La robótica Swarm o en español Robótica de Enjambre es una nueva aproximación a la coordinación de un gran numero de robots relativamente simples. De forma que estos mismos robots sean capaces de de llevar a cabo tareas colectivas que están fuera de las capacidades de un único robot [5].

Se supone que un comportamiento colectivo deseado surge de las interacciones entre los robots y las interacciones de los robots con el entorno. Este enfoque surgió en el campo de la inteligencia artificial de enjambres, así como en los estudios biológicos de insectos, hormigas y otros campos en la naturaleza [5].

Este campo de investigación tiene justamente su inspiración en el comportamiento observado en los insectos sociales, en los que se destacan, las hormigas, termitas, abejas o avispas; los cuales son ejemplos de como un gran numero de individuos simples pueden interactuar para crear sistemas inteligentes colectivos [5].

6.2. Ventajas de la Robótica Swarm

Los sistemas robóticos de enjambre son tolerantes a fallos y robustos, ya que pueden continuar con su misión ante el fallo de alguna unidad. Aprovechan al máximo el paralelismo ya que el conjunto de robots ejecuta mas rápido cualquier tarea que un único robot, descomponiendo la tarea en subtareas y ejecútenlas de manera concurrente. Además que estos robots son bastante mas baratos y el coste de cualquier reparación también es bastante menor en comparación con cualquier gran robot [5].

6.3. Particle Swarm Optimization PSO

El algoritmo de Optimización del enjambre de partículas (PSO) fue creado por por el Dr. Russell Eberhart y el Dr. James Kennedy en el año de 1995. Este tiene origen la simulación de de comportamientos sociales utilizando herramientas e ideas tomadas por gráficos computa rizados e investigación sobre psicología social; además de una clara inspiración en el comportamiento social de las crías de aves y la educación de los peces [6] .

El algoritmo PSO pertenece a las técnicas denominadas optimización inteligente y se clasifica como un algoritmo estocástico de optimización basado en población. A esta clasificación igualmente pertenecen los Algoritmos Genéticos (AG). PSO es intrínsecamente paralelo. La mayoría de algoritmos clásicos operan secuencialmente y pueden explorar el espacio de solución solamente en una dirección a la vez [6].

Este algoritmo es iniciado y simula un grupo aleatorio de partículas a las cuales se les asigna una posición y velocidad inicial, a estas partículas se les conoce como "soluciones", para luego proceder a actualizar las generaciones de estas para encontrar la solución óptima. En cada iteración cada partícula es actualizada por los siguientes 2 mejores resultados [7].

El primero de estos se le conoce como la mejor solución lograda hasta ahora por cada partícula y recibe el nombre de *local best*. El segundo mejor valor es rastreado por el PSO y este proceso se repite hasta que se cumpla con un número de iteraciones especifica o se logre la convergencia del algoritmo. La convergencia se alcanza cuando todas las partículas son atraídas a la partícula con la mejor solución la cual recibe el nombre de *global best* [7]. La figura No. 2 representa una ejemplificación de como los varios *local best* deben de converger a un único *global best*.

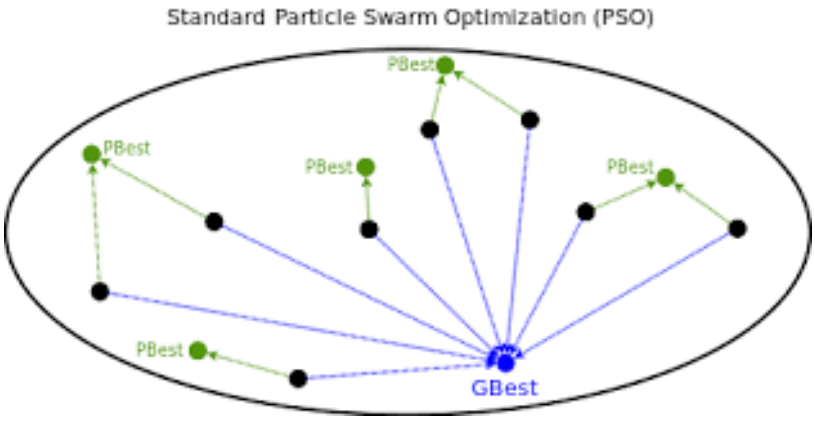

Figura 2: Ejemplo de convergencia [7]

Con estos datos es posible calcular los dos primeros factores principales de la ecuación del *PSO* para cada una de las partículas, los cuales son:

- Factor Cognitivo: $P_{local} - P_i$
- Factor Social: $P_{global} - P_i$

Donde P_i representa cada una de las soluciones/posiciones actuales. Para verificar que tan buena es la posición actual para cada partícula se usa la denominada *funcion fitness* que se da por $f(P)$ [1], el valor escalar que genera como resultado se le denomina costo el objetivo de las partículas es encontrar un conjunto de coordenadas que generen el valor de costo más pequeño posible dentro de una región dada.

Se toma en cuenta una diversidad de factores, entre ellos, el factor de escalamiento el cual consiste en hacer que las partículas tengan una mayor área de búsqueda o hacer que se concentren en un área mas reducida, definidas como $C1$ y $C2$. El factor de uniformidad son 2 numeros aleatorios que van entre 0 y 1; se definen como $r1$ y $r2$. El factor de constricción φ el cual se encarga de controlar la velocidad de cada partícula [1]. El factor de inercia w el cual se encarga de controlar cuanta memoria puede almacenar cada partícula. De forma que podemos armar la ecuación de velocidad brindada por el PSO.

$$V_{i+1} = \varphi[wV_i + C1r1(P_{local} - P_i) + C2r2(P_{global} - P_i)] \tag{1}$$

Donde V_i representa la velocidad actual de la partícula y V_{i+1} la nueva velocidad calculada para la partícula. Una vez actualizadas las velocidades de todas las partículas,se calcula las posiciones de cada una de estas:

$$X_{i+1} = X_i + V_{i+1} * \Delta t \tag{2}$$

Donde X_i representa la posición actual de la partícula y X_{i+1} la nueva posición calculada para la partícula, Δt es el tiempo que le toma al algoritmo realizar cada iteración. En la figura No. 3 vemos un ejemplo de los diferentes parámetros y como estos tienen efectos sobre cada una de las partículas presentes en el algoritmo.

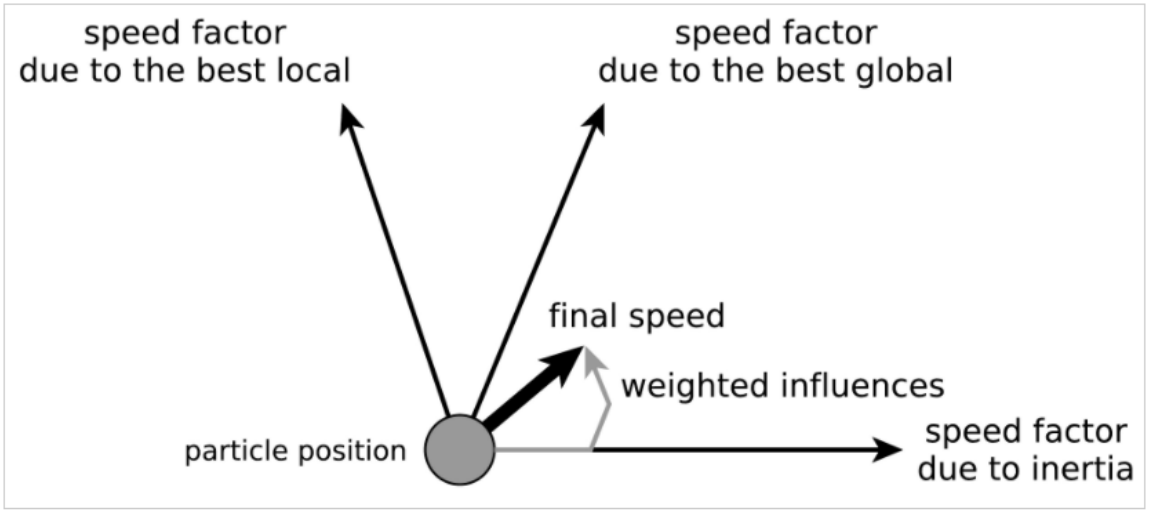

Figura 3: Efectos sobre la partícula [6].

6.4. Funciones Costo o Fitness

Para verificar el correcto funcionamiento del algoritmo PSO se utilizan varias funciones las cuales contiene múltiples mínimos y máximos, al ser tan usadas para validacion de este algoritmo se les conoce como *benchmark functions* o *Fitness functions*. Al usar estas funciones se tiene una mejor idea si los parámetros usados en el algoritmos PSO son correctos o no, dentro de las evaluaciones tenemos, velocidad de convergencia ,capacidad de identificar mínimos o máximos, si la función presenta varios mínimos locales el algoritmos debe ser capaz de identificarlos y converger en el mínimo global.

6.4.1. Función Sphere

Es una de las funciones mas usadas, en su forma bidimensional es continua y convexa, no posee mínimos locales únicamente un minino global ubicado en [0,0] [8].

$$f(x) = \sum_{i=1}^{N} x_i^2 \tag{3}$$

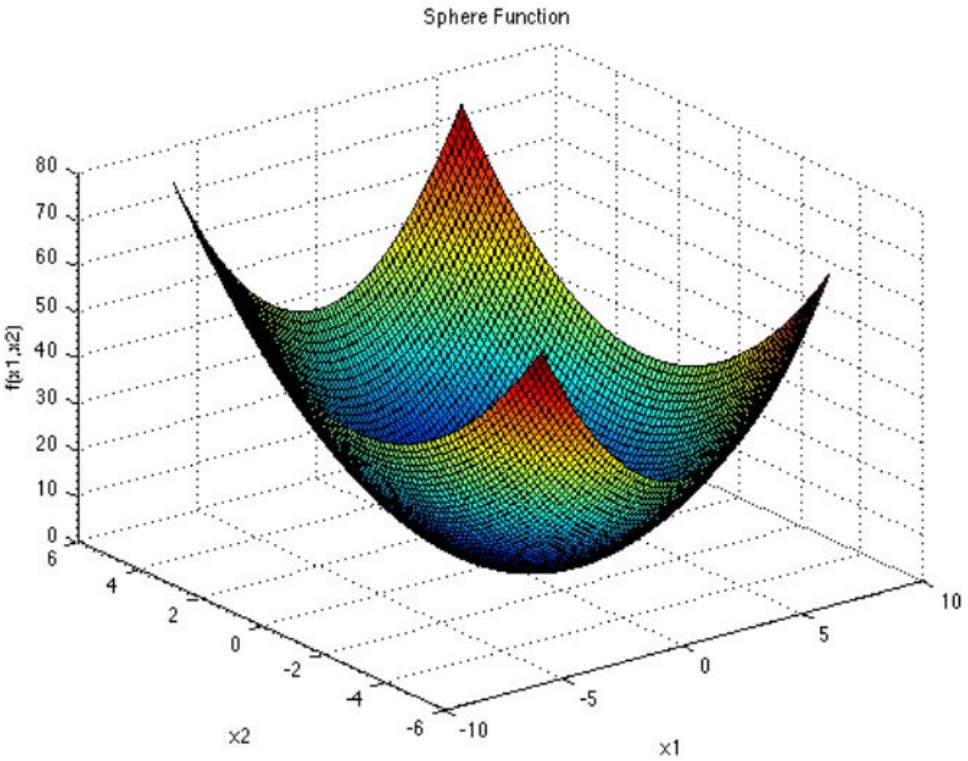

Figura 4: Representación de la función Sphere [8].

6.4.2. Función Rosenbrock

Este tipo de función en su forma bidimensional es continua y posee un único mínimo global ubicado en [1,1] [9].

$$f(x) = \sum_{i=1}^{N-1} (1 - x_i)^2 + 100(x_{i+1} - x_i^2) \tag{4}$$

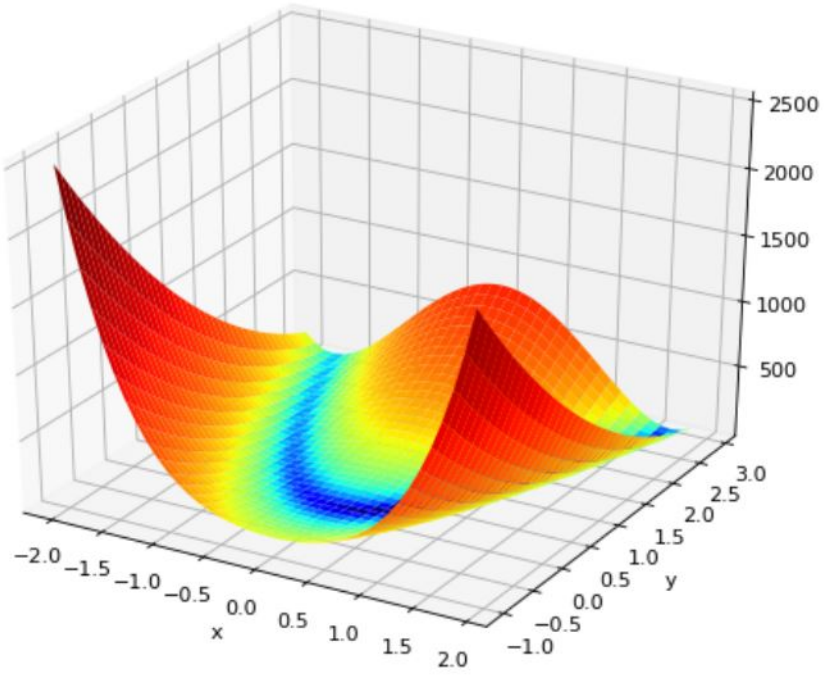

Figura 5: Representación de la función Rosenbrock [9].

6.4.3. Función Himmelblau

Este tipo de función se caracteriza por tener múltiples mínimos del mismo valor, este hace que el algoritmo detecte múltiples mínimos globales. Es útil para determinar que tanto afecta la posición inicial de las partículas o agentes [9].

$$f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \tag{5}$$

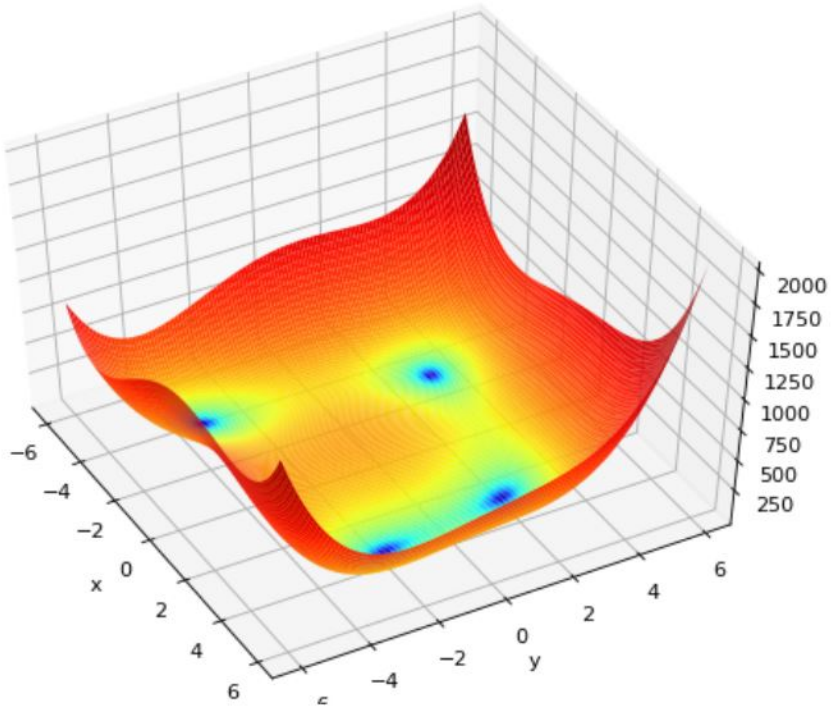

Figura 6: Representación de la función Himmelblau [9].

6.5. Programación Multihilos

En la programación Multihilos se hace referencia a los lenguajes de programación que permiten la ejecución de varias tareas de forma simultanea. Los hilos o *threads*, son pequeños procesos o piezas independientes de un gran proceso; de igual forma se puede decir, que un hilo es un flujo único de ejecución dentro de un proceso [10].

Un hilo no puede correr por si mismo, se ejecuta dentro de un programa, ya que requieren la supervisión de un proceso padre o un *main thread* para correr. Se pueden programar múltiples hilos de ejecución para que se corran simultáneamente en el mismo programa [10].

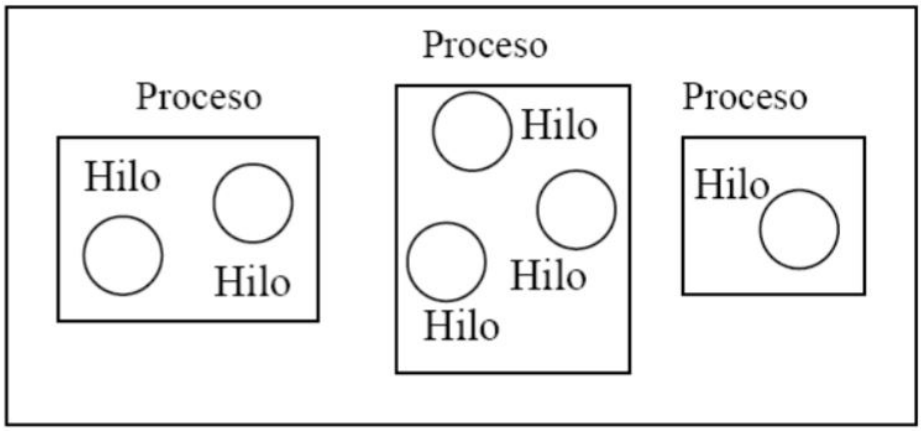

Figura 7: Ejemplo de los programas o *main thread* encargados de correr los diferentes hilos [10].

6.5.1. Casos en los que conviene usa programación Multihillos

El hecho de usar programación multihilos no siempre implica que vamos a tener una mejora de rendimiento y es decisión del programador cuándo conviene usar este tipo de programación. En términos generales, si se tiene una aplicación muy simple no tiene sentido plantearse este tipo de programación [11].

Es común el uso de programación multihilos cuando se desea realizar diversas tareas claramente diferenciadas con un alto coste computacional, se debe tener en cuenta que el resultado de las tareas que se ejecuten en diferentes hilos no deben depender del resultado de otras tareas ya que esto crea una dependencia que haría el proceso menos eficiente. El caso mas común es cuando se prevea que pueda haber tareas retenidas o bloqueadas por estar esperando algún tipo de señal de activación [11].

Otra ventaja importante es el ahorro de tiempo al usar programación multihilos, ya que se tendrán diferentes programas ejecutándose a la vez y no se tendrá que esperar a que uno se termine para comenzar el siguiente proceso, como normalmente ocurre. Como se puede ver en la Figura No. 5 se tiene un *Main Thread* del cual se crean 4 *Thread* o 4 hilos cada uno ejecutando un programa diferente del otro, se ilustra el tiempo de ejecución de cada hilo y el programa principal tarda 20 segundos, siendo este el tiempo del *Thread* 3, el mas largo, por lo que se tiene un ahorro de tiempo importante en comparación a ejecutar los 3 hilos de forma secuencial.

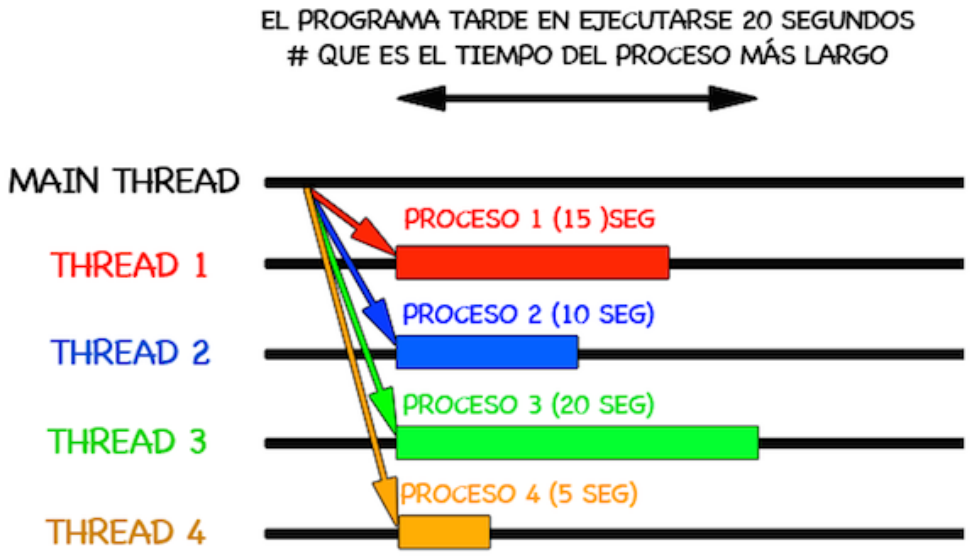

Figura 8: Ejemplo de un proceso Multihilos [11].

6.6. Raspberry Pi

Se le conoce como Raspberry Pi a la serie de ordenadores de placa reducida desarrollados por el Reino Unido por la *Raspberry Pi Foundation*. Los modelos Raspberry fueron creados para la enseñanza de la informática en escuelas, a medida que su éxito creció se desarrollaron modelos más complejos dando lugar diversas versiones lo que dio lugar a la *Raspberry Pi Trading* encargada de la producción de las nuevas versiones [12].

6.6.1. Hardware

Raspberry Pi utiliza un arquitectura para el procesador ARM. Esta arquitectura es de tipo RISC (*Reduced Instruction Set Computer*), es decir, utiliza un sistema de instrucciones realmente simple lo que le permite ejecutar tareas con un mínimo consumo de energía [12].

6.6.2. Software

Raspberry Pi OS es el sistema operativo recomendado para el uso común en una Raspberry Pi, este sistema operativo es de uso gratuito basado en Debian, optimizado para el hardware Raspberry Pi. Este sistema operativo cuenta con mas de 35,000 paquetes y se puede descargar desde la pagina oficial: Raspberry Pi OS, se descarga una copia exacta del sistema operativo en una Ïmagen", la cual contiene la estructura y los contenidos completos de un sistema operativo para luego ser copiada en una tarjeta SD para su uso en el hardware Raspberry Pi, el proceso de copiar la imagen en una SD se realiza mediante la aplicación gratuita *Raspberry Pi Imager* [13].

6.6.3. Modelos

Con forme el paso del tiempo la compañía *Raspberry Pi Trading* fue desarrollando diferentes modelos de acuerdo a las nuevas tecnologías disponibles, entre los cuales podemos encontrar:

- Raspberry Pi 1 modelo A (descontinuada)
- Raspberry Pi 1 modelo B (descontinuada)
- Raspberry Pi 2 modelo B
- Raspberry Pi 3 modelo B+
- Raspberry Pi 3 modelo A+
- Raspberry Pi 4 modelo B

A parte de los modelos normales, también se han desarrollado otra gama de placas denominadas Raspberry Pi Zero. Estas son mucho más pequeñas y menos potentes que sus hermanas, pero es precisamente su atractivo, menos gasto y un precio mucho menor [14].

6.7. Robots Móviles

La robótica móvil se considera actualmente un área de la tecnología avanzada manejadora de problemas de alta complejidad. Las propiedades características de los robots son la versatilidad y la autoadaptabilidad. La primera se entiende como la potencialidad estructural de ejecutar tareas diversas, lo cual implica una estructura mecánica de geometría variable. La autoadaptabilidad significa que un robot debe, por sí solo, alcanzar su objetivo a pesar de las perturbaciones imprevistas del entorno a lo largo de la ejecución de su tarea [15].

El proceso más básico para la Navegación de un robot móvil se basa en el modelo cinemático del sistema de propulsión. Este sistema es el que permite al robot moverse dentro de un determinado entorno. Uno de los sistemas más usuales se basa en el uso de ruedas de tracción diferencial [16].

6.7.1. Modelo Uniciclo

El modelo uniciclo consiste en una sola rueda y relaciona la velocidad angular de la rueda con su eje de rotación paralela al suelo y la velocidad lineal de la rueda con respecto al piso. Esto permite plantear las velocidades lineales de las ruedas como:

$$V_R = \phi_R r \tag{6}$$

$$V_L = \phi_L r \tag{7}$$

$$w = w \tag{8}$$

Donde la velocidad angular de ambos motores es ϕ, r es el radio de las ruedas y w es la velocidad angular del robot.

6.7.2. Modelo Diferencial

El modelo diferencial de un robot móvil considera ya 2 ruedas, se relacionan las velocidades lineales con el centro de masa del robot, al tener un robot uniforme la velocidad lineal del centro de masa es el promedio de las velocidades lineales de cada llanta, de forma que podemos plantear las ecuaciones de velocidad de ambas llantas en función de la velocidad lineal y angular.

$$V_R = \frac{v + wl}{r} \tag{9}$$

$$V_L = \frac{v - wl}{r} \tag{10}$$

Donde l es el radio del robot y r el radio de las llantas.

6.7.3. Difeomorfismo para la transformación de cinemática uniciclo

Debido a que el sistema dinámico de un robot diferencial es no lineal el aplicarle control no es una buena idea. Por lo que necesitamos realizar un ajuste llamado difeomorfismo buscando reducir la información que tenemos del robot móvil, con el difeomorfismo buscamos controlar el robot diferencial conociendo la velocidad y orientación de un punto [17].

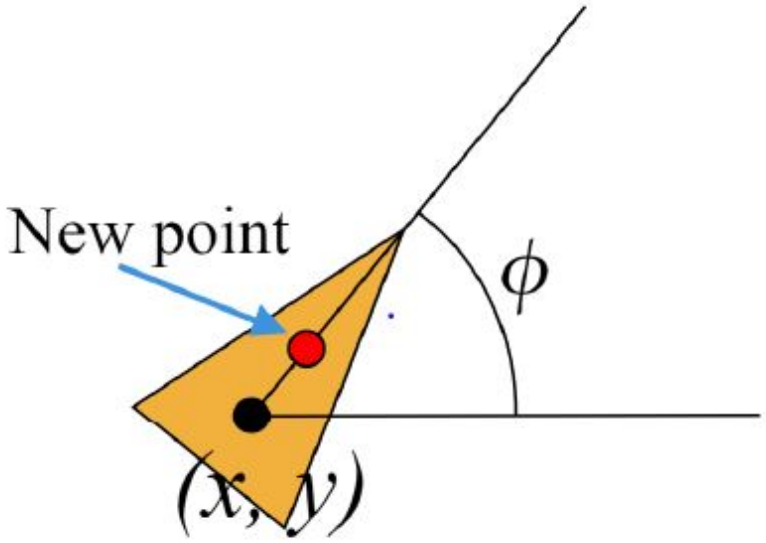

Figura 9: Ajuste del difeomorfismo [17].

Gracias a este ajuste es posible plantear las ecuaciones de velocidad lineal y angular, para ver el desarrollo algebraico completo se recomienda leer el capitulo 6 de [1] así como sus referencias.

$$v = u_1 \cos\phi + u_2 \sin\phi \tag{11}$$

$$w = \frac{-u_1 \sin\phi + u_2 \cos\phi}{l} \tag{12}$$

Donde u_1 y u_2 representan el vector de control y l la distancia entre el centro y el punto de difeomorfismo.

6.8. Controladores de Posición y Velocidad de Robots Diferenciales

Estos controladores se plantean para asegurar que el robot logre llegar a la posición deseada, además que la trayectoria seguida sea suave, para ver un mayor desarrollo en estos controladores se recomienda leer el capitulo 6 de [1].

6.8.1. Control proporcional de velocidades con saturación limitada

Este controlador usa los errores de posición, orientación y las velocidad para acotar las velocidades del robot para ejecutar la trayectoria dada [16]. Las entradas de control se definen de la forma:

$$u_1 = I_x \tanh \frac{k_x(x_g - x)}{I_x} \tag{13}$$

$$u_2 = I_y \tanh \frac{k_y(y_g - y)}{I_y} \tag{14}$$

Donde (x_g y y_g) representan la meta y (x y y) representan la posición actual, I_x y I_y son las constantes de saturación y se encargan de evitar que las velocidades resultantes sean demasiado grandes cuando el error inicial de posición sea demasiado grande. Al aplicar estas entradas de control con el difeomorfisto tenemos:

$$v = I_x \tanh \frac{k_y(y_g - y)}{I_y} \cos\phi + I_y \tanh \frac{k_y(y_g - y)}{I_y} \sin\phi \tag{15}$$

$$w = \frac{-I_x \tanh \frac{k_x(x_g - x)}{I_x} \sin\phi + I_y \tanh \frac{k_y(y_g - y)}{I_y} \cos\phi}{l} \tag{16}$$

6.8.2. Control PID de velocidad lineal y angular

En los robots móviles uno de los controladores mas usados es el PID, parte primordial de este controlador es la determinación de la constantes K_P, K_I y K_D, si bien existen ciertas normas para la determinación de estas constantes calcularlas al tanteo resulta una buena practica. Con estas constantes se busca reducir el error en estado estable así como mejorar los parámetros de rendimiento t_p, t_s y M_p, la ecuación general del controlador PID viene dada por:

$$PID(e(t)) = K_P e(t) + K_I \int_0^t e(t)dt + K_D \frac{de(t)}{dt} \tag{17}$$

Su implementación para la velocidad lineal y angular esta dada por el uso del error de orientación e_o y el error de posición e_p, estos son calculados:

$$e_o = atan2(\frac{\sin(\theta_g - \theta)}{\cos(\theta_g - \theta)}) \tag{18}$$

$$e_p = \sqrt{(x_g - x)^2 + (y_g - y)^2} \tag{19}$$

Donde θ_g es al ángulo calculado desde el punto actual hasta la meta y θ ángulo actual.

6.8.3. Control de pose

El control de pose toma en consideración la pose u orientación final que tendrá el robot al llegar a la meta, esta pose final depende la pose inicial que tenga el robot. Para profundizar en el desarrollo algebraico se recomienda leer el capitulo 6 de [1]. Las ecuaciones polares de coordenadas son:

$$\rho = \sqrt{(\Delta x)^2 + (\Delta y)^2} \tag{20}$$

$$\alpha = -\theta + atan2(\Delta y, \Delta x) \tag{21}$$

$$\beta = -\theta - \alpha \tag{22}$$

Mientras que las leyes de controlador de pose son:

$$v = k_p \rho \tag{23}$$

$$w = k_\alpha \alpha k_\beta \beta \tag{24}$$

Donde ρ es el error de posición este ayuda al robot a orientarse y seguir la linea entre el robot y la meta, mientras β se encarga de orientar dicha línea asegurándose que el robot logre la pose final al llegar a la meta.

6.8.4. Control de pose de Lyapunov

Este controlador toma como base el criterio de estabilidad de Lyapunov para asegurar sea asintóticamente estable. Este criterio nos dice que si existe una solución lo suficientemente cerca de un punto de equilibrio X_o de una ecuación diferencial homogénea, esta se mantiene cerca para todo $t > t_o$, de igual forma este punto de equilibrio X_o es asintóticamente estable si posee estabilidad de Lyapunov además de ser un atractor de soluciones. Para un mayor desarrollo y desarrollo algebraico se recomienda leer el capitulo 6 de [1].

$$v = k_p \rho \cos\alpha \tag{25}$$

$$w = k_p \sin\alpha \cos\alpha k_\alpha \alpha \tag{26}$$

6.8.5. Control Closed-Loop Steering

Este tipo de control de pose es de caracter similar a los anteriores variando en su calculo de velocidad angular, para profundizar en este tipo de controlador se recomienda leer el capitulo 6 de [1].

$$v = k_p \rho \cos\alpha \tag{27}$$

$$w = \frac{2v}{5\rho}(k_2(\alpha + atan(-k_1\beta)) + (1 + \frac{k_1}{1 + (k_1\beta)^2})\sin\alpha) \tag{28}$$

6.8.6. Control por medio de Regulador Lineal Cuadrático (LQR)

El control LQR permite la estabilización de un sistema dinámico alrededor de un punto de operación, de igual forma este busca realizar control sobre un sistema de forma óptima, esto quiere decir usar la menor cantidad de control posible. Para un mayor detalle del desarrollo matemático se recomienda leer el capitulo 6 de [1]. Recordemos que el sistema de un robot diferencial es no lineal usamos el difeomorfismo para aplicar el control LQR, de forma que tenemos:

$$u = -K(x_g - x) + u_g \tag{29}$$

6.8.7. Controlador Lineal Cuadrático Integral (LQI)

Debido a que el control LQR no es robusto contra las perturbaciones, y existe cierta incertidumbre entre el modelo del sistema y el sistema real, se busca compensar el error en estado estable agregando una parte integral . Recordemos que el sistema de un robot diferencial es no lineal usamos el difeomorfismo para aplicar el control LQI, de forma que tenemos:

$$u = -K_1 x + K_2 \sigma \tag{30}$$

6.9. Propuesta de Robots

6.9.1. Kilobot

El Kilobot es un sistema robótico de bajo costo, estos enjambres están inspirados en insectos sociales, como colonias de hormigas, que pueden buscar y encontrar eficientemente fuentes de alimento en grandes ambientes complejos, transportar colectivamente grandes objetos y coordinar la construcción de nidos y puentes en tales ambientes [18].

El Kilobot está diseñado para proporcionar a los científicos un banco de pruebas físicas para avanzar en la comprensión del comportamiento colectivo y realizar su potencial para ofrecer soluciones para una amplia gama de desafíos. Este tiene un máximo de 33mm de diámetro [18].

Algunas de sus características físicas son:

- Diámetro: 33 mm
- Altura: 34 mm
- Costo: 12 dolares
- Max. distancia de comunicación : 7 cm
- Autonomía: 1 horas en movimiento

Figura 10: Kilobot [18].

6.9.2. E-puck

Este robot fue desarrollado por la Escuela Politécnica Federal de Lausana (EPFL) el cual fue creado para fines educativos y de investigación. Este pequeño robot es cilíndrico y cuenta con dos ruedas, equipado con una variedad de sensores cuya movilidad está garantizada por un sistema de accionamiento diferencial. Tanto su diseño como librerías y manuales de uso son open source y todo se encuentra disponible en su pagina oficial [19].

Algunas de sus características físicas son:

- Diámetro: 70mm
- Altura: 50 mm
- Peso: 200 g
- Max. velocidad: 13 cm/s
- Autonomía: 2 horas en movimiento

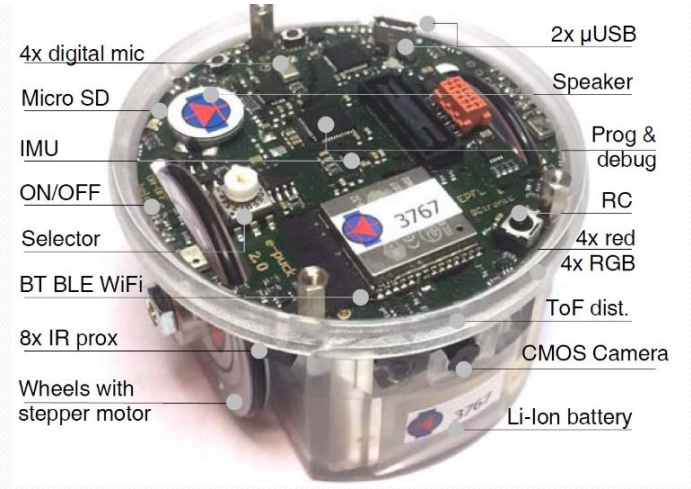

Figura 11: E-puck [19].

6.9.3. Pi-puck

El Pi-puck es una plataforma robótica desarrollada por el *York Robotics Laboratory* en la Universidad de York y GCtronic. Este básicamente es una extensión del e-puck y el e-puck2, el cual posee integrado una Raspberry Pi Zero agregando soporte Linux así como periféricos adicionales. Al ser de uso comercial se cuenta con una distribución de software YRL para el Pi-Puck en donde se incluye una imagen personalizada de Raspian y varios paquetes que permiten un completo control sobre el hardware y una serie de bibliotecas ya instaladas para hacer mas fácil el uso del robot [20].

La Raspberry Pi Zero se conecta al robot por medio de I2C, posee un micrófono digital y un altavoz. Consta te 2 baterías internas las cuales se cargan por medio de cable USB. Posee seis canales de I2C, dos entradas ADC así como múltiples leds para verificar funciones de encendido.

El precio de cada pi-puck es de Q 3,163.89 los cuales son distribuidos en Guatemala por *GCtronic*

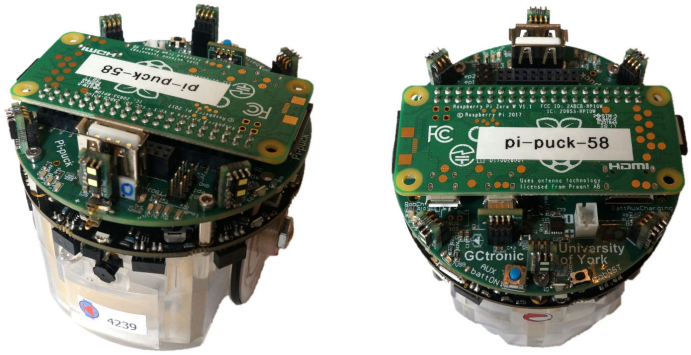

Figura 12: E-puck [20].

CAPÍTULO 7

Implementación del Particle Swarm Optimization

Para la implementación del algoritmo PSO en un sistema físico se toma como base los repositorios elaborados en las fases previas a esta tesis, como es mencionado en la sección de antecedentes. Se toma en consideración la obtención de hiperparámetros calculados haciendo uso de redes neuronales donde se obtienen los mejores resultados a nivel de simulación.

7.1. Raspberry Pi y Lenguaje C

Para realizar la correcta migración hacia un sistema físico se evaluó los diferentes tipos de robots móviles que actualmente se encuentran en el mercado así como sus respectivos precios, como parte de otro proyecto de tesis actualmente se trabaja en el desarrollo de una plataforma móvil pensada para el uso de robótica de enjambre. El ordenador *Raspberry* es elegido gracias a su afinidad con los robots móviles, más específicamente con el *Pi-puck* como se describe en la sección del marco teórico y posteriormente en la nueva plataforma móvil UVG.

Otros microcontroladores pensados para esta implementación fueron Arduino Uno, PIC 16f877a y Tiva-C, una clara desventaja de estos microcontroladores es su futura implementación con robots móviles. Además sus claras limitaciones en espacio de memoria al ser comparados con un ordenador como la *Raspberry*.

Otra ventaja al usar el modelo *Raspberry* es la previa experiencia que se tiene con este ordenador gracias a cursos previos y que la adquisición de este no fuera un problema tanto para mi como para la Universidad ya que se requieren de varios modelos para realizar la validación del algoritmo PSO. El modelo seleccionado para la realización de las pruebas es *Raspberry Pi 4*, versión la cual ya poseo. Mientras que la Universidad cuanta con el modelo *3B+* con el cual no varían demasiado en procesador o en la memoria RAM por lo que se

ajusta bastante bien a la implementación que se le dará.

Figura 13: Raspberry Pi 4 [12].

Se opta por el uso del lenguaje tipo C ya que de igual forma se busca un lenguaje con el cual se tenga una previa experiencia y la comodidad de ya haberlo usado. Ya que *Raspberry* no tiene un entorno de desarrollo exclusivo, se opta por trabajar en Visual Studio Code, haciendo uso de su propia terminal para compilar y posteriormente ejecutar las pruebas del algoritmo. Se aprovechan herramientas de este editor de código como el resaltado de sintaxis y su finalización de código inteligente. Además este es fácil de instalar y es bastante amigable al usuario brindando facilidad en temas de edición.

Figura 14: Visual Studio Code [12]

7.2. Metodología

Para la implementación del PSO se quiere que los hiperparámetros, constantes de controladores y diferentes tipos de inercia así como funciones *fitness* sean fácil de modificar, de forma que se declaran como constantes. Así mismo se crean los espacios de memoria para el almacenamiento de los diferentes parámetros internos del PSO y los que serán usados para la interpretación del GPS y los encargados del envió y recepción de información hacia otros agentes, que en este caso son representados por otros modelos *Raspberry*.

7.2.1. Creación de funciones fitness

Se procedió a crear las diferentes funciones *benchmark* usadas normalmente para evaluar el correcto funcionamiento del PSO, estas previamente descritas en el marco teórico, las funciones *fitness* creadas fueron: *Sphere*, *Rosenbrock* y *Himmelblau*.

El proceso de evaluación consistió en evaluar la posición de cada agente (X,Y) en la función *fitness* seleccionada. Este resultado es usado para determinar el costo de posición de cada agente. Durante la implementación se trabajara con 3 tipos de costo, los cuales son:

- Fitness actual: esta evaluación toma en consideración la posición actual de cada agente.
- Fitness local: esta evaluación toma en consideración la posición inicial de cada agente y es el costo que se comunica a los demás agentes. Este valor se actualiza si el *fitness actual* posee un menor costo.
- Fitness global: esta evaluación toma en consideración la posición inicial de cada agente y solo se actualiza si el *fitness local* posee un menor costo (el propio o el enviado por otro agente).

7.2.2. Programación multihilos

Como se menciono en secciones previas y posteriormente sera mejor descrito en el capitulo 8, se hace uso de ciertas estrategias para validar el funcionamiento del PSO debido a la falta de una plataforma o robot móvil. Para estas estrategias se hace uso de la programación multihilos para que estos puedan ser programas ejecutados al mismo tiempo que el algoritmo PSO, cabe mencionar que todas las variables declaradas en los hilos creados únicamente se pueden usar en dichos hilos, para evitar esto todas las variables son declaradas como globales, evitando así este inconveniente.

Para la creación de los hilos se hace uso de la función pthread_create(), la cual inicial un hilo que comienza a ejecutarse en paralelo al momento de ser utilizada, como argumento de la función se hace uso de una variable tipo thread para cada hilo y se llama a la función que debe ejecutar cada hilo (envió y recepción de datos y lectura del GPS) en la Figura 15 vemos un diagrama ejemplificando este proceso. Finalmente se usa pthread_exit() para cerrar el hilo. Para que todo esto pueda funcionar se hace uso de la librería *pthread.h*

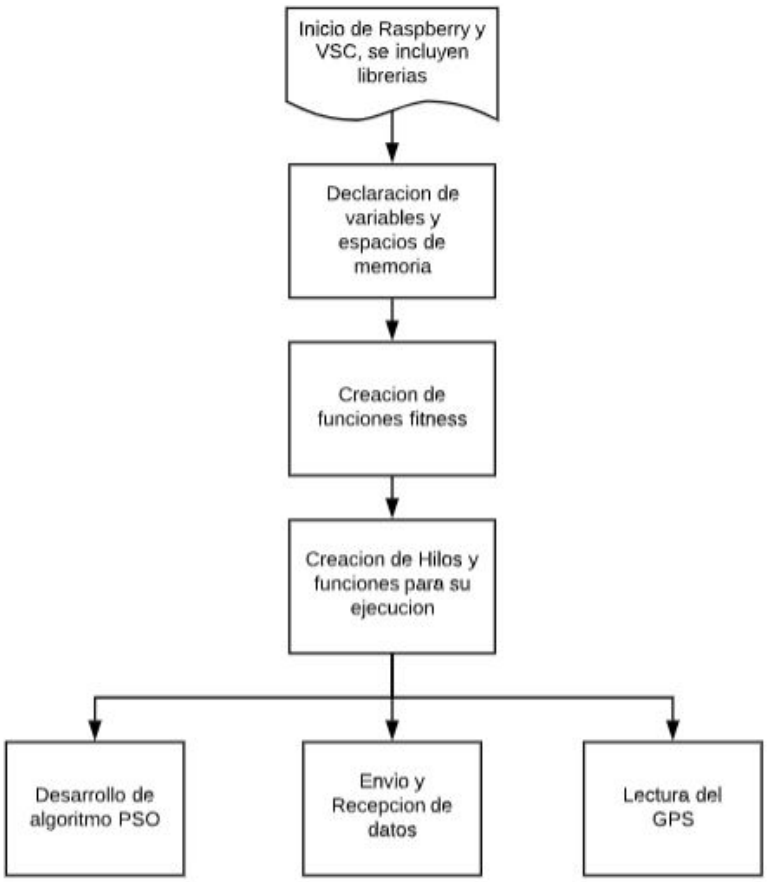

Figura 15: Generación de Hilos.

7.2.3. Configuración inicial del PSO

Esta configuración se hace al inicio del Main Loop y solo ocurre una vez, tiene como objetivo registrar la posición inicial de cada agente y asignar este valor al *best local* y *best global* así como evaluar la posición inicial para el calculo del costo del *fitness local* y *fitness global*.

7.2.4. Obtención de posición y orientación del robot

Con la posición inicial ya establecida, se registra la posición y orientación actual, este tema sera ampliado en el capitulo 8, cabe resaltar que durante la implementación del código PSO modificado de la fase II [1], se tuvo que hacer un arreglo para la orientación debido a problemas con la brújula interna del robot usado en Weboots. Esto no fue necesario con el modulo GPS el cual sera explicado en el capitulo 8.

7.2.5. Actualización del local y global best

Con la posición actual registrada se calcula el costo del *finess actual*, el diagrama visto en la Figura 16 ejemplifica este proceso, cabe resaltar que los cuadros en amarillo representan su ejecución en otro hilo de programación.

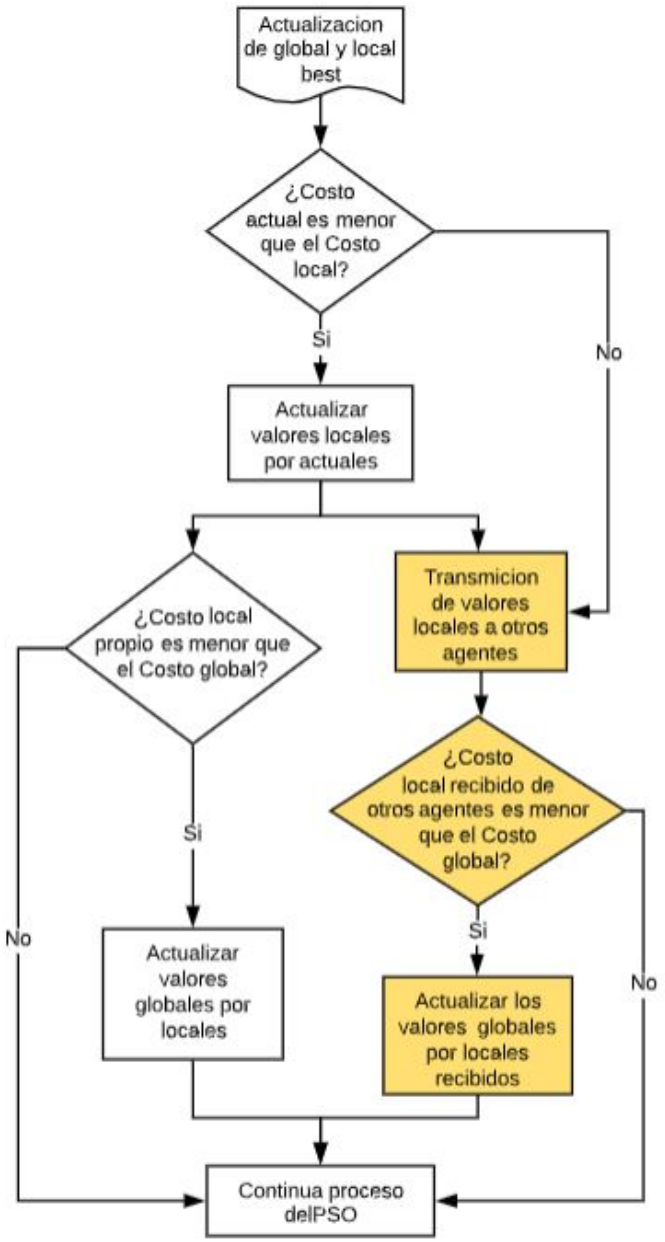

Figura 16: Actualización del local y global best.

7.2.6. Algoritmo PSO

Para el calculo de parámetros de uniformidad, las ecuaciones del tipo de inercia a utilizar, el parámetros de constricción y el valor del V_Scaler, se hace uso de las ecuaciones y repositorios de github elaborados en la fase II [1], si se desea una mayor descripción ver repositorios adjuntos de github. Se realiza el calculo para una nueva velocidad, esta ecuación es previamente descrita en el marco teórico, ecuación (1). De igual forma se procede con la realización del calculo de la nueva posición de cada agente ecuación (2) descrita en el marco teórico.

Adicionalmente se hace el calculo del error de posición para las cordenas X y Y:

$$e = X_{i+1} - X_i \tag{31}$$

7.2.7. Calculo de la velocidad lineal y velocidad angular mediante varios tipos de controladores

Para el calculo de ambas velocidades se hace uso de los distintos tipos de controladores descritos en el capitulo No. 6, así como los usados en los repositorios de la fase II [1]. En estos mismos encontramos las ecuaciones que son usadas en la programación así como los parámetros que se toman.

Los controladores usados son:

- Control proporcional de velocidades con saturación limitada
- Control PID de velocidad lineal y angular
- Control de pose
- Control de pose de Lyapunov
- Control Closed-Loop Steering
- Control por medio de Regulador Lineal Cuadrático (LQR)
- Controlador Lineal Cuadrático Integral (LQI)

La selección del tipo de controlador a usar se realiza al inicio del código.

7.2.8. Envió de velocidades a los motores

Se hace uso del modelo diferencial y las ecuaciones (9 y 10) para enviar las velocidades a los respectivos motores. Sin embargo como se pretende limitar las velocidades al máximo posible por los motores del robot físico se hace uso de la variable *MAX_SPEED* a la cual se le asigna el valor que puede dar el motor usado en el E-puck (este valor se modificara dependiendo de los motores usados en la plataforma móvil en desarrollo), este proceso se describe en la Figura 17.

Figura 17: Proceso de truncar variables de velocidad.

Ambas velocidades son desplegadas en la ventada de comandos para darles una interpretación en espera de la plataforma móvil.

CAPÍTULO 8

Consideraciones del PSO

Debido a que no se tiene una plataforma móvil en la cual validar el funcionamiento del PSO, actualmente se está trabajando en el desarrollo de un robot móvil integrado con una *Raspberry*, se buscan métodos alternos para realizar esta validación.

8.1. Procesamiento de la data

Parte fundamental del correcto funcionamiento del PSO es el poder comunicar la posición actual así como el fitness actual a los otros agentes para tomar una decisión sobre el recorrido a tomar. En una implementación donde se cuente con algún robot móvil como el *Pi-puck* o el *E-puck* se podría hacer uso de su sistema ya integrado de comunicación para la transmisión de la data, sin embargo como esta plataforma aún está en desarrollo se busca una alternativa diferente pero eficiente para realizar él envío y recepción de la data.

8.1.1. Envío y Recepción de la información

Para la validación del PSO se creó una red local haciendo uso de un router al cual se conectan todos los agentes o modelos *Raspberry*, se hizo uso del modelo $4B$ y $3B$. Para él envío y recepción de data se hace uso de la técnica de *broadcast* en la cual todos los agentes conectados a la red pueden mandar información a la vez y de igual forma todos pueden recibir esta información, siempre y cuando todos estén conectados al mismo puerto.

Como se puede ver en la Figura 18 los N cantidad de agentes se conectan a la misma red local y al mismo puerto, de esta forma todos son capaces de enviar y recibir información al mismo tiempo, para facilidad de interpretación se crearon 2 variables, una para él envío *buffer enviar* y otra para la recepción *buffer recibir*.

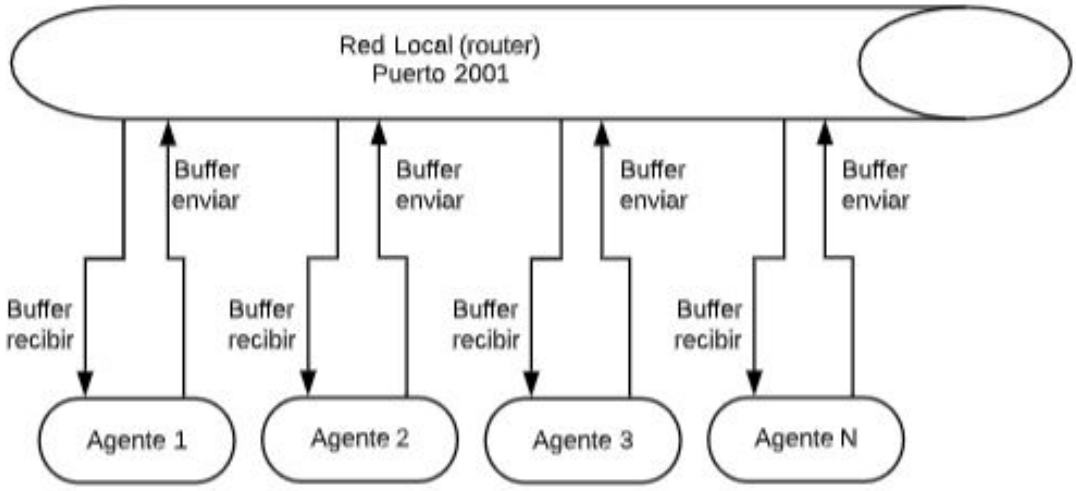

Figura 18: Representación del envió y recepción de datos.

Para el proceso de envío y recepción de la data se hace uso de programación multihilos, se realiza de este modo ya que el proceso de recibir información hace uso de una función de bloqueó, la cual detiene el flujo del programa hasta haber llenado *buffer* de datos o la variable *buffer recibir*, con programación multihilos se crea un hilo que se encarga de ejecutar la función *receiving* de forma que no interrumpa el resto del código.

```
pthread_create(&hilo_envio, NULL, receiving, (void *)&sock); // para recepcion
```

El único proceso ejecutado en este hilo es el de recibir la información de otros agentes y comparar el costo global con el costo local recibido por otros agentes, el proceso de enviar información se ejecuta en el programa principal.

8.1.2. Ordenamiento de la información

La información de interés se ordena de forma conveniente antes de ser enviada, para esto se hace uso de la función *sprintf* la cual permite almacenar la información dentro de una variable, en este caso se almacena en la variable *buffer enviar*. Esta información es separa por medio de comas. La información de interés que se envía son los valores locales tanto de posición (X,Y) como el costo local que se tiene por agente.

```
sprintf(buffer_enviar,"\%f,\%f,\%f", best_local[0],best_local[1],fitness_local);
```

Esta información se recibe por medio de la función *receiving* (ya en otro hilo de programación) y se procede a separar la información que se encuentra dentro del *buffer*, para esto se hace uso de la función *strtok* la cual se encarga de realizar la separación cada vez que

encuentre una coma, posteriormente se hace uso de la función *atof()* encargada de convertir una cadena en un valor numérico, esto es necesario ya que el uso de la función *sprintff* devuelve una cadena de caracteres. Como se menciono en el capitulo 7, estas variables son usadas nuevamente en el programa principal por lo que son declaradas como globales, de lo contrario únicamente podrían ser usadas en el hilo creado para el proceso de recibir información.

En la Figura 19 se puede ejemplificar este proceso, cabe resaltar que los cuadros en amarillo representan su ejecución en otro hilo de programación.

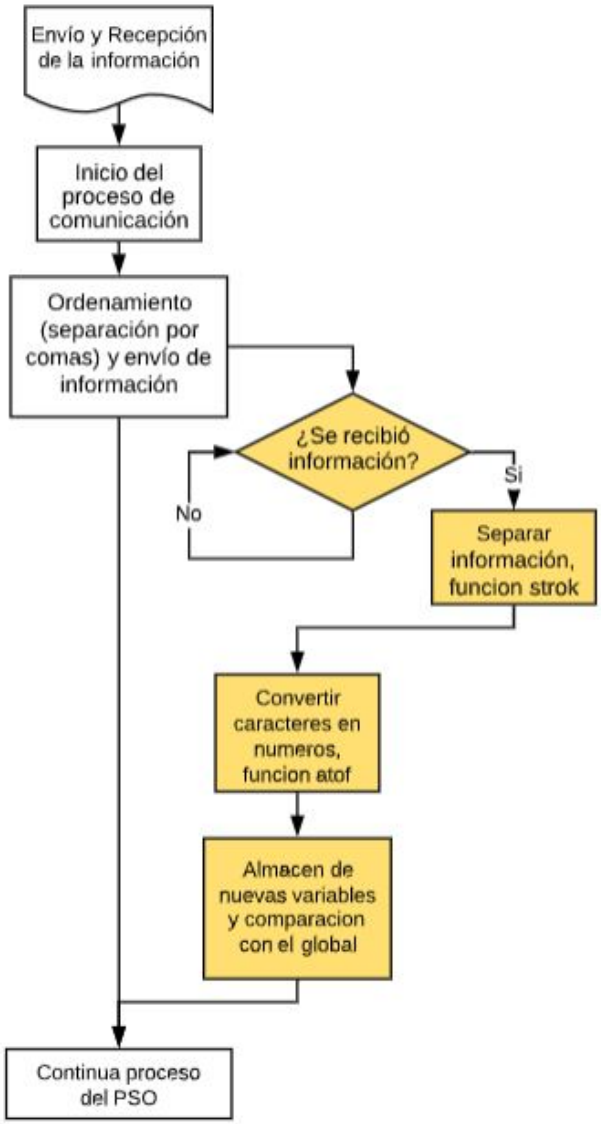

Figura 19: Diagrama de envió y recepción de dato.

8.1.3. Validación del Proceso

Para las pruebas del envío y recepción de la data, así como la prueba del uso de programación multihilos, se elaboró un código sencillo, el cual tiene como objetivo agrupar la información introducida manualmente (los valores locales) para su envió por medio del puerto seleccionado; esta información luego de ser recibida es separada y si se cumple con tener un mejor *fitness* se actualiza el valor global. Se hizo uso de 4 agentes con 4 valores locales diferentes, estos resultados se pueden en el Cuadro 1.

Agente	fitness local inicial	fitness global inicial	fitness global final
1	10.0	10.0	2.0
2	6.0	6.0	2.0
3	4.0	4.0	2.0
4	2.0	2.0	2.0

Cuadro 1: Ambos fitness iniciales muestran los valores introducidos manualmente, mientras que los finales muestran el cambio realizado con la comparación

Para lograr visualizar el cambio, se imprime en la consola si se logra actualizar el valor global o no. El objetivo de la prueba es que todos los agentes envíen su posición actual y que todos sean capaces tanto de recibirlos como de actualizar sus valores globales si se detecta un mejor *fitness local*. Cabe mencionar que él envió se realiza a la misma velocidad programada por defecto en el router.

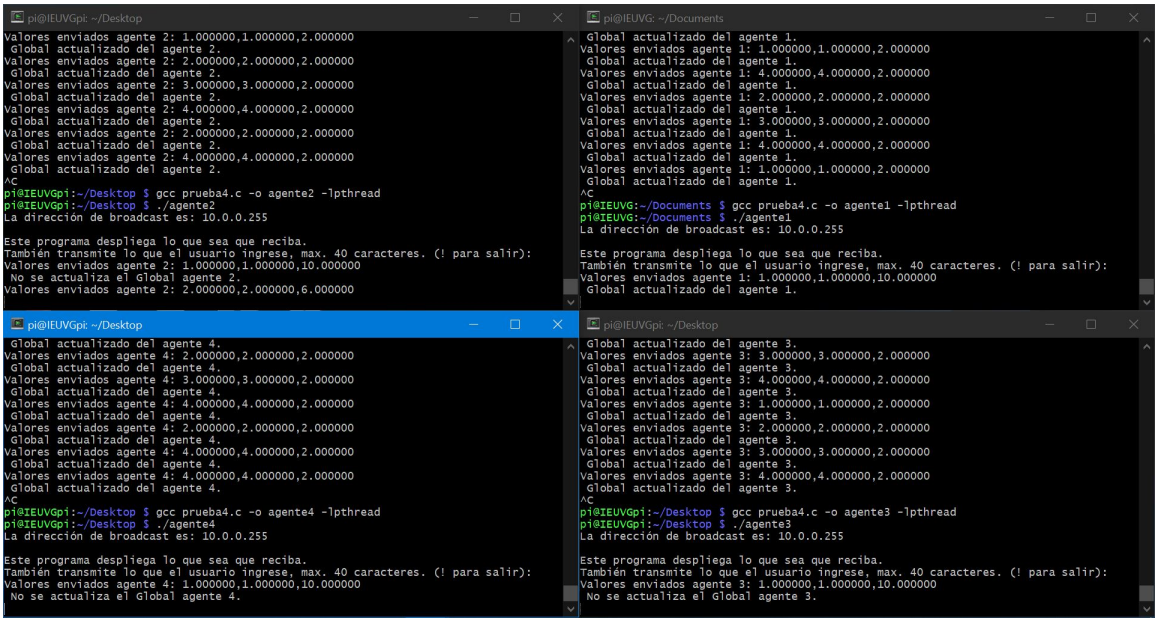

Figura 20: Pruebas del envió y recepción de datos 1.

En la Figura 20 se puede ver que los 4 agentes envían su posición local, sin embargo, ya que los programas fueron ejecutados de forma secuencial solo el agente 1 ha sido capaz de actualizar su valor global.

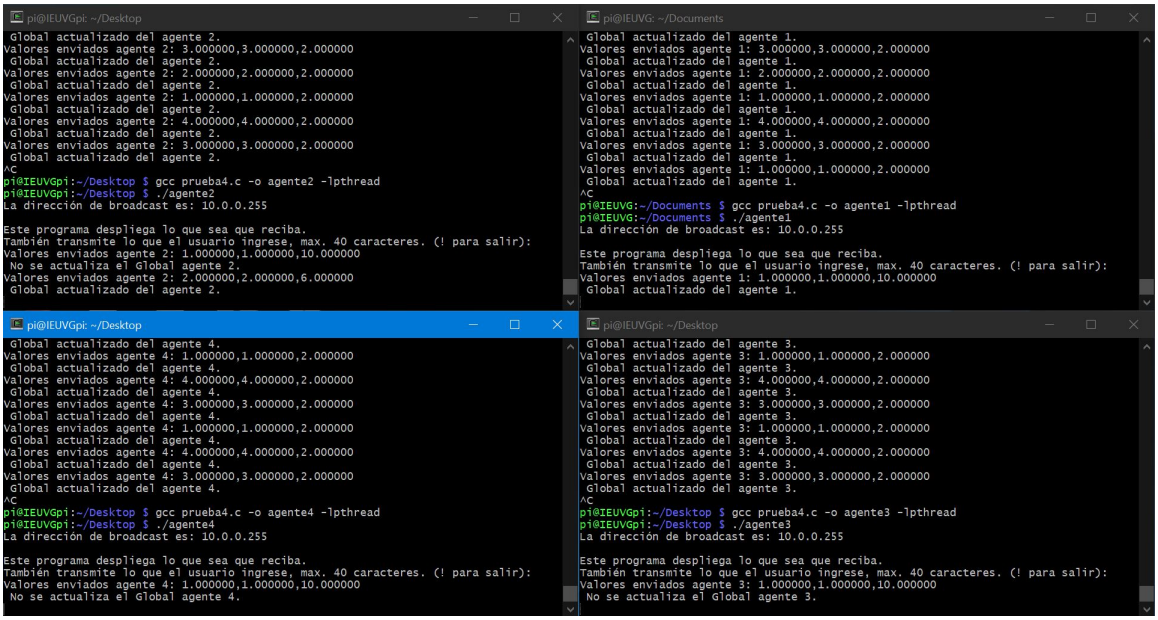

Figura 21: Pruebas del envió y recepción de datos 2.

En la Figura 21 al haber transcurrido un mayor tiempo el segundo agente ya actualiza su posición global, mientras todos los demás agentes continúan enviando su posición actual por medio del *buffer*.

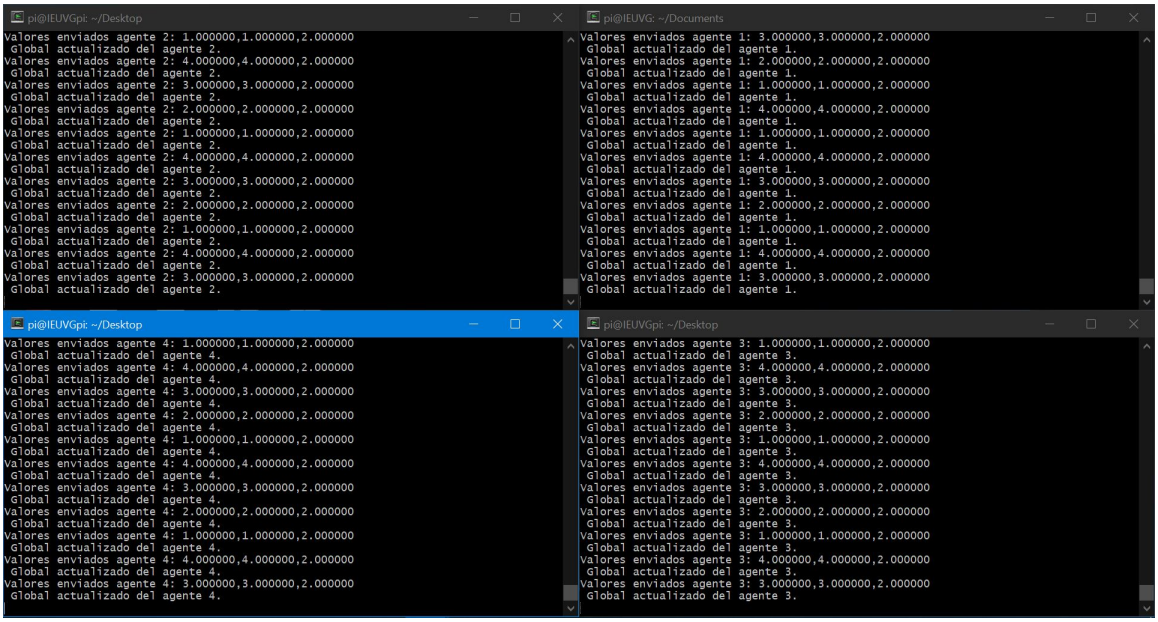

Figura 22: Pruebas del envió y recepción de datos 3.

Finalmente, en la Figura 22 ya con un mayor tiempo transcurrido se puede ver cómo los 4 agentes logran actualizar su posición global, ya que el mejor *fitness* fue el del agente 4 los demás son capaces de reconocer esto y modificar su propio valor global.

8.2. Obtención de Coordenadas

Otra parte fundamental del funcionamiento del algoritmo PSO es que cada agente sea inicializado con una posición inicial y que este sea capaz identificar sus nuevas posiciones ya iniciado el algoritmo, en aplicaciones únicamente simuladas, la posición inicial es dada aleatoriamente o haciendo uso de entornos como webots se usa el *GPS* interno del robot utilizado.

8.2.1. Módulo GPS

Buscando una aplicación más realista del PSO se opta por el uso de un módulo GPS que permita obtener las coordenadas X y Y (latitud y longitud) en tiempo real, este es el *Adafruit Ultimate GPS Breakout*.

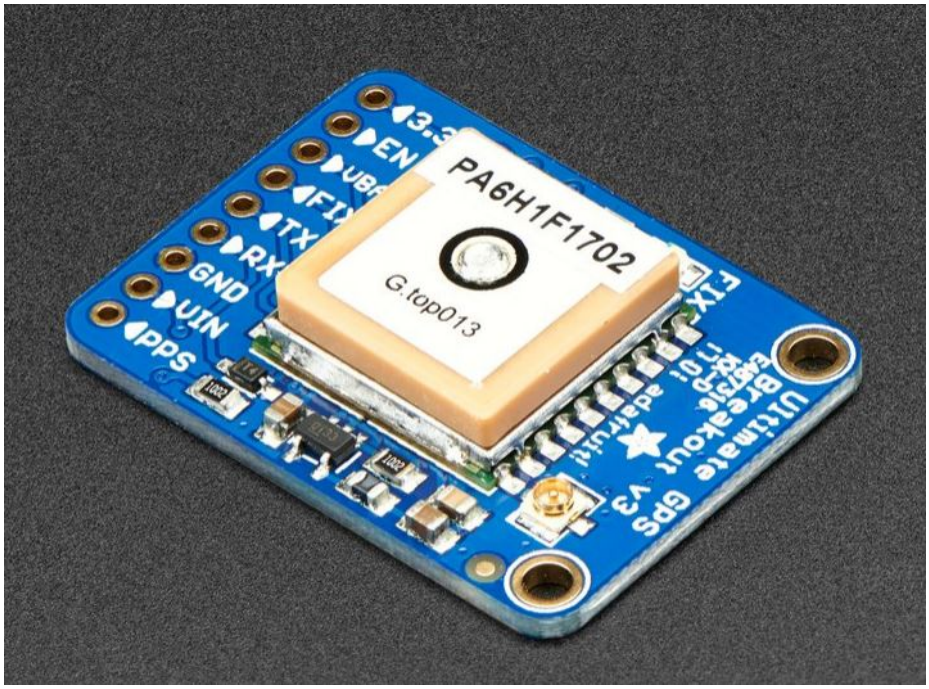

Figura 23: Módulo Adafruit Ultimate GPS Breakout [21].

Este módulo se conecta a la *Raspberry* por comunicación serial y funciona con 3.3 o 5V, en la Figura 24 se muestra un diagrama de la conexión del módulo GPS.

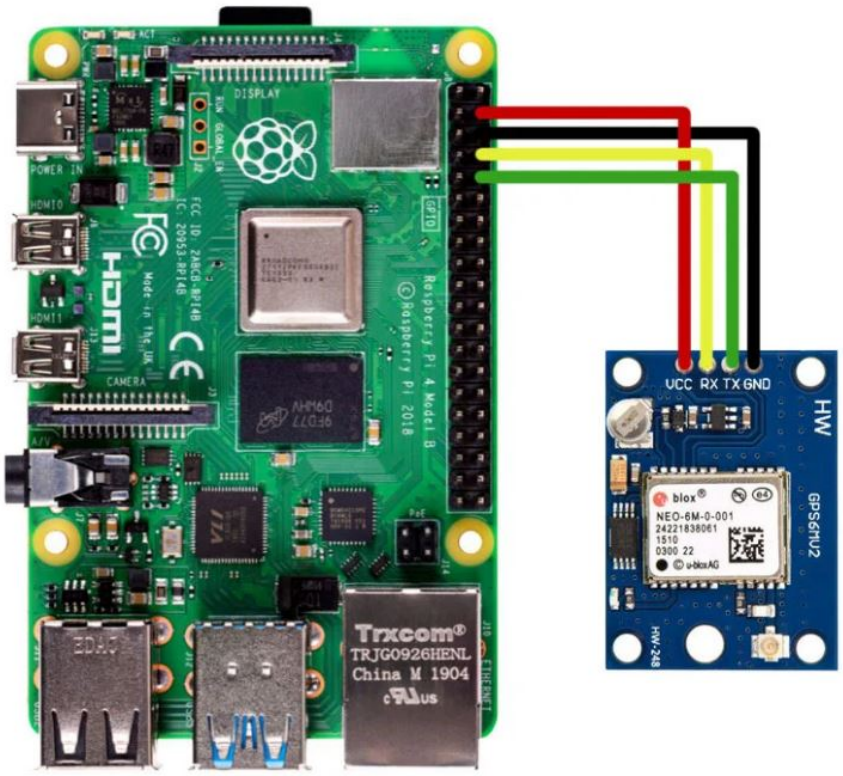

Figura 24: Conexión del GPS [22].

Como primer paso se comprobó el correcto funcionamiento del módulo GPS habilitando la interfaz serial y deshabilitando el *login shell* y por medio de una serie de comandos mejor descritos en [22] se creó una ventana de visualización, la cual se puede ver en las Figuras 25 y 26.

Figura 25: Prueba del modulo GPS 1.

Como es mencionado en [22] las primeras veces que se inicia el modulo no responde de la mejor forma como se puede ver en al Figura 25; pasado un tiempo de aproximadamente 1 hora el módulo se logra conectar y comienza a desplegar la información de interés.

Esto se puede ver en la Figura 26, los datos de longitud y latitud son comparados con el GPS de mi teléfono haciendo uso de la aplicación Google Earth.

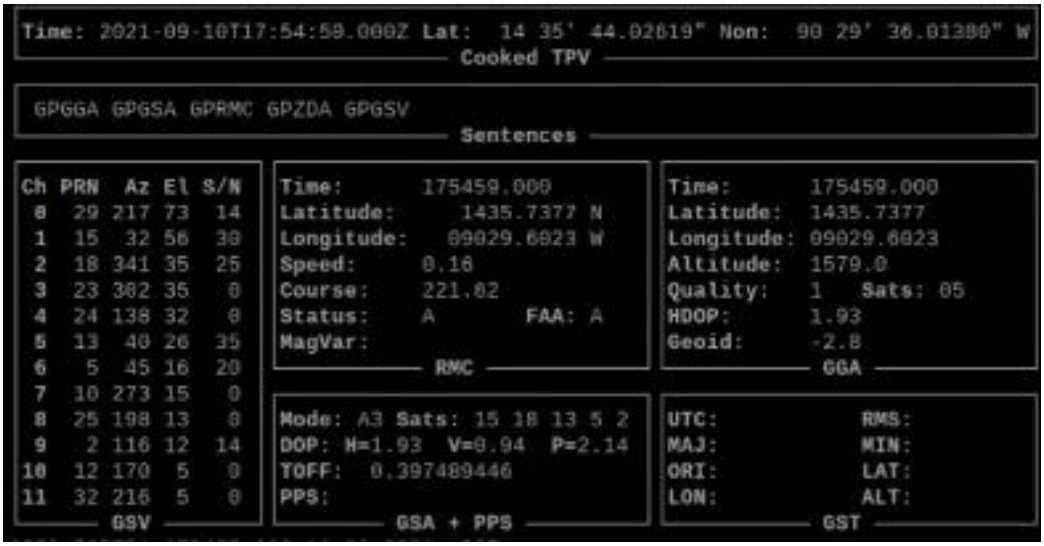

Figura 26: Prueba del modulo GPS 2.

Para uso del algoritmo PSO son necesarios 3 datos, la longitud, la latitud y la orientación. Para la extracción de estos datos se hace uso de la librería *gps.h*, para una explicación mas profunda de su uso se recomienda leer [23] donde se explica como instalarla y los comandos de ejecución necesarios para su uso.

8.2.2. Lectura del módulo

El proceso de lectura del módulo GPS se realiza haciendo uso de programación multihilos, esto se hace de esta forma ya que el proceso de lectura se mantiene bloqueado mientras no se logre establecer una conexión entre el modulo y el satélite en uso. Para evitar que esto interrumpa el flujo del programa principal se crea un hilo encargado almacenar la información deseada para su uso en el PSO mas adelante.

```
pthread_create(&hilo_gps, NULL, gps, NULL); // para GPS
```

Figura 27: Hilo de lectura del GPS.

8.3. Dimensiones del robot

Como se menciona en el capitulo 7, las dimensiones del robot son de vital importancia para el calculo de las velocidades lineal y angular por medio de los controladores y como la futura plataforma móvil aun se encuentra en desarrollo se usan las dimensiones de otro robot móvil, el E-puck.

Estas dimensiones son asignadas a variables para ser usadas en el código, de esta forma el proceso de cambiarlas es bastante simple y al momento que la plataforma móvil se encuentre operativa se podrán agregar sus dimensiones y realizar las pruebas de validación.

CAPÍTULO 9

Validación por medio de pruebas físicas

9.1. Mesa de pruebas UVG

9.2. Otros agentes Usados

9.3. Validación del PSO en sistemas físicos

9.4. Comparación con resultados a nivel de simulación

9.5. Interpretación de resultados

CAPÍTULO 10

Conclusiones

- El módulo GPS funciona de forma óptima luego de pasar por el proceso de iniciación, es compara con la aplicación de Google Earth.
- La velocidad de envió de datos es adecuada para su uso en el algoritmo PSO.
- La creación de varios hilos ayuda a mantener un mejor orden en el código, otra ventaja de la programación multihilos.
- Los parámetros de inercia, constricción y escalamiento dieron un buen resultado a nivel de simulación por lo que se usan para comparar la simulación con la implementación física.

CAPÍTULO 11

Recomendaciones

- Se recomienda hacer la implementación con algún otro tipo de robot móvil y compararlo con el obtenido con la Raspberry.
- Se recomienda probar otro tipo de comunicación entre los agentes, aparte de utilizar una red local con un router.
- En la implementación en la plataforma móvil se recomienda usar el mismo módulo GPS que ya fue validado.

CAPÍTULO 12

Bibliografía

[1] A. S. A. Nadalini, "Algoritmo Modificado de Optimización de Enjambre de Partículas (MPSO)," Tesis de licenciatura, Universidad del Valle de Guatemala, 2019.

[2] E. A. S. Olivet, "Aprendizaje Reforzado y Aprendizaje Profundo en Aplicaciones de Robótica de Enjambre," Tesis de licenciatura, Universidad del Valle de Guatemala, 2020.

[3] Jason Maderer, *Robotarium: A Robotics Lab Accessible to All*, `https://www.news.gatech.edu/features/robotarium-robotics-lab-accessible-all`, Accessed: 2021-03-27, 2017.

[4] G. I. Colmenares, "Aprendizaje Automático, Computación Evolutiva e Inteligencia de Enjambre para Aplicaciones de Robótica," Tesis de licenciatura, Universidad del Valle de Guatemala, 2020.

[5] L. S. Tortosa, "Ajentes y enjambres artificiales: modelado y comportamientos para sistemas de enjambre robóticos," Tesis doct., Universidad de Alicante, España, 2013.

[6] R. F. R. Grandi y C. Melchiorri, *A Navigation Strategy for Multi-Robot Systems Based on Particle Swarm Optimization Techniques*. Dubrovnik, Croatia: Dubrovnik, 2012.

[7] C. Duarte y C. J. Quiroga, *PSO algorithm*. Ciudad Universitaria, Santander, Colombia: Santander, 2010.

[8] D. Bingham and S. Surjanovic, *Virtual Library of Simulation Experiment*, `https://www.sfu.ca/~ssurjano/spheref.html`, Accessed: 2021-08-13, 2013.

[9] D. Bingham, *Project Homepage DEAP*, `https://deap.readthedocs.io/en/master/api/benchmarks.html#deap.benchmarks.sphere`, Accessed: 2021-09-13, 2021.

[10] C. CABALLERO GONZÁLEZ, *Programación con lenguajes de guión en páginas web*. S.A.: Ediciones Paraninfo, 2015.

[11] J. Lajara y J. Pelegri, *LabVIEW: Entorno gráfico de programación*. S.A.: Marcombo, 2011.

[12] Raspberry Pi, *Raspberry Pi*, `https://www.raspberrypi.org`, Accessed: 2021-06-04, 2021.

[13] ——, *Raspberry Pi OS*, https://www.raspberrypi.org/software/, Accessed: 2021-06-04, 2021.

[14] ——, *Raspberry Pi Products*, https://www.raspberrypi.org/products/, Accessed: 2021-06-04, 2021.

[15] R. MÓVILES, *G. BERMÚDEZ*, Universidad Distrital Francisco José de Caldas.Colombia, dic. de 2001.

[16] A. M. J. Valencia y L. Rios, *MODELO CINEMÁTICO DE UN ROBOT MÓVIL TIPO DIFERENCIAL Y NAVEGACIÓN A PARTIR DE LA ESTIMACIÓN ODOMÉTRICA*, Universidad Tecnológica de Pereira, mayo de 2009.

[17] M. Egerstedt, *Control of Mobile Robots, Introduction to Controls*, Georgia Institute of Technology, mayo de 2014.

[18] K-Team, *KILOBOT*, https://www.k-team.com/mobile-robotics-products/kilobot, Accessed: 2021-02-25, 2017.

[19] GCtronic, *e-puck education robot*, http://www.e-puck.org, Accessed: 2021-03-27, 2018.

[20] University of York, *Pi-puck*, https://pi-puck.readthedocs.io/en/latest/, Accessed: 2021-06-04, 2020.

[21] Adafruit, *Adafruit Ultimate GPS Breakout*, https://www.adafruit.com/product/746?gclid=Cj0KCQjw1ouKBhC5ARIsAHXNMI_os3oMgM-8lXhcOW_yNP8xjWwhfgtck4VbwjLg-WSiLwL4-vOko7QaAkcwEALw_wcB, Accessed: 2021-09-05, 2021.

[22] Pro Maker, *How to Use a GPS Receiver With Raspberry Pi 4*, https://maker.pro/raspberry-pi/tutorial/how-to-use-a-gps-receiver-with-raspberry-pi-4, Accessed: 2021-09-10, 2021.

[23] Walter Dal Mut, *LibreriaGPS*, https://github.com/wdalmut/libgps, Accessed: 2021-09-10, 2021.

CAPÍTULO 13

Anexos

13.1. Código

Creación de funciones Costo:

```
// Funciones costo
double funcion(double x, double y){
    double f=0;
    if (funcion_costo==0){
        f=pow(x,2)+pow(y,2);                          // Sphere
    }else if (funcion_costo==1){
        f=pow(1-x,2)+100*pow(y-pow(x,2),2);           // Rosenbrock
    }else if (funcion_costo==2){
        f=pow(x+2*y-7,2)+pow(2*x+y-5,2);              // Booth
    }else if(funcion_costo==3){
        f=pow(x*x+y-11,2)+pow(x+y*y-7,2);             // Himmelblau
    }
    return f;
}
```

Creación de función receiving:

```
void *receiving(void *ptr)
{
    int *sock, n;
    int i = 0;
    sock = (int *)ptr;                                  // socket identifier
    unsigned int length = sizeof(struct sockaddr_in); // size of structure
    struct sockaddr_in from;
```

```
    while (1){
        memset(buffer_recibir, 0, MSG_SIZE); // "limpia" el buffer
        // receive message
        n = recvfrom(*sock, buffer_recibir, MSG_SIZE, 0,
        (struct sockaddr *)&from, &length);
        if (n < 0){
            error("Error: recvfrom");
        }
        i = 0;
        //------ Se descompone el buffer_recibir---------------------
        token = strtok(buffer_recibir, ",");
        recepcion[i] = atof(token);
        while ((token = strtok(NULL, ",")) != NULL){
            i++;
            recepcion[i] = atof(token);
        }
        // ---------------Actualizar global best ----------------------
        if (recepcion[2] < fitness_global){
            printf("Global actualizado.\n");
            best_global[0] = recepcion[0];
            best_global[1] = recepcion[1];
            fitness_global = recepcion[2];
        }
        else{
            printf("No se actualiza el Global.\n");
        }
    }
    pthread_exit(0);
}
```

Creación de función gps:

```
void *gps(void *ptr){
    // Open the communication
    gps_init();
    loc_t data;
    while (1) {
        gps_location(&data);
        // Visualizacion de la informacion
        printf("\%lf \%lf\n", data.latitude, data.longitude);
        posicion_robot[2]= data.longitude;
        posicion_robot[0]= data.latitude;
        rad = data.course;
    }
    pthread_exit(0);
}
```

CAPÍTULO 14

Glosario

Google Earth Google Earth es una app que permite explorar un globo terráqueo virtual, elaborado a partir de datos cartográficos e imágenes satelitales.. 37

memoria RAM Es la memoria principal de un dispositivo, esa donde se almacenan de forma temporal los datos de los programas que estás utilizando en este momento.. 23

Visual Studio Code Visual Studio Code es un editor de código fuente que permite trabajar con diversos lenguajes de programación, admite gestionar tus propios atajos de teclado y refactorizar el código.. 24

Webots Es un simulador de robots 3D gratuito y de código abierto, usado para simular robots móviles se suele utilizar con fines educativos.. 3