Taller 3 Robótica

Juan David Medina, Jorge Felipe Gaviria, Maria Alejandra Escalante, Juan David Serrano
Universidad de los Andes

I. Introducción

En este taller se busca resolver distintos problemas haciendo uso de uno o más nodos de ROS. Se deben implementar algoritmos para encontrar rutas como BFS para el juego de Pacman, el algoritmo A estrella y Rapidly exploring Random Trees para encontrar rutas dados obstáculos en el robot Pioneer y en un último punto, representar la información entregada por un sensor laser acerca de los obstáculos en el mapa.

II. Herramientas utilizadas

Se deben instalar algunas librerías de la siguiente manera:

- sudo pip install –user networkx
- sudo -H python -m pip install –upgrade pip setuptools wheel.
- sudo -H pip install xlib
- sudo -H pip install pynput
-

III. Punto 1

III-A. Análisis del algoritmo

Para correr el primer punto se debe usar el siguiente comando (con el mapa mediumCorners ya abierto):
rosrun taller3_4 punto1a.py
Para solucionar este problema se uso el siguiente algoritmo:

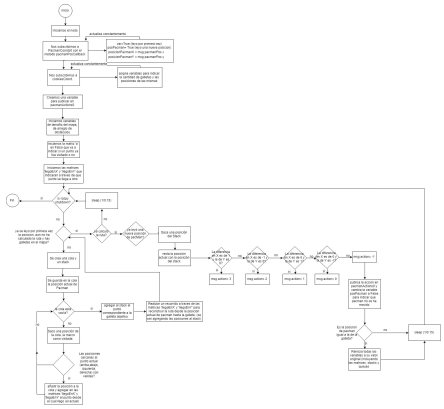


Figura 1: Algoritmo Punto 1a

El algoritmo se puede ver ampliado en los anexos.
A continuación presentamos un ejemplo o representación visual del funcionamiento del algoritmo
Primero asumimos un mapa como el de la figura 2 donde hay un Pacman y una galleta a la que se debe llegar. Tenemos 2 tablas. La primera indica a través de que punto llegue a ese y la otra indica si un punto ya fue visitado o no.

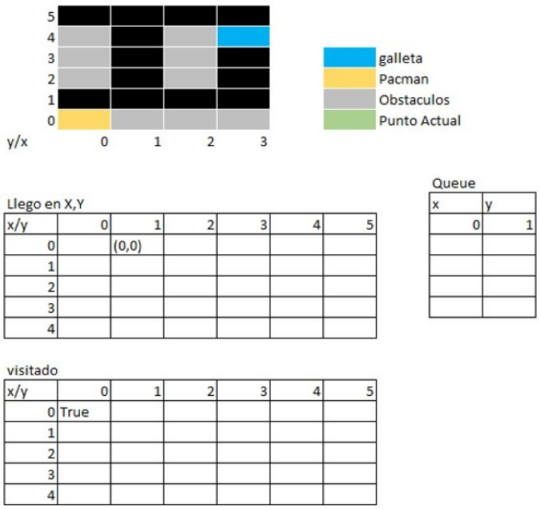


Figura 2: Ejemplo paso 1

Realizando el algoritmo de BFS vamos llenando las tablas expandiendo los puntos usando una Queue.

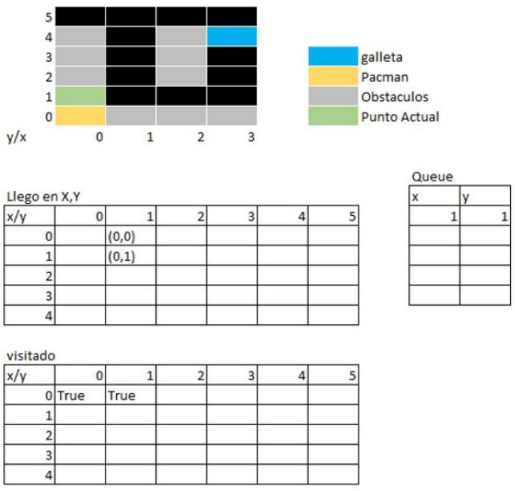
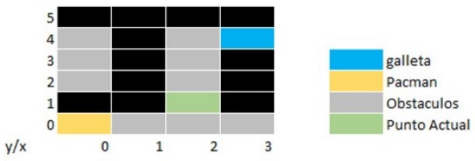


Figura 3: Ejemplo Paso 2

Al terminar el algoritmo se obtiene lo siguiente:



Llego en X,Y

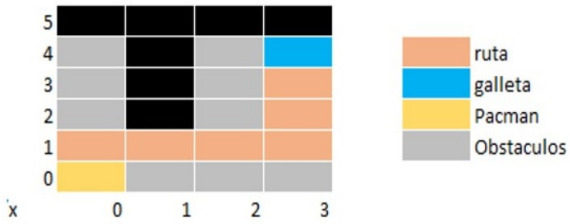
x/y	0	1	2	3	4	5
0	(0,0)	(0,0)				
1		(0,1)	(1,1)	(1,2)	(1,3)	(1,4)
2		(1,1)				(1,5)
3		(2,1)	(3,1)	(3,2)	(3,3)	
4						

visitado

x/y	0	1	2	3	4	5
0	True	True				
1		True	True	True	True	
2		True				
3		True	True	True	True	
4						

Figura 4: Ejemplo Terminado

Analizando las tablas podemos crear un Stack para reconstruir la ruta mas corta que nos llevo desde Pacman hasta la galleta.



stack ruta
(0,1)
(1,1)
(2,1)
(3,1)
(3,2)
(3,3)

Figura 5: Ejemplo- Reconstrucción de ruta

Sabemos que es la ruta mas corta, ya que el algoritmo se detiene al llegar a la galleta, es decir, no se sigue expandiendo por caminos mas largos. Debido a que el Stack funciona de forma LIFO (Last in First out) podemos sacar cada uno de los puntos en orden y definir los movimientos de Pacman.
El tiempo de muestreo de los nodos involucrados fue elegido por el tiempo en el que se demoran en actualizarse PacmanCoords y CookieCoord y se creo una variable booleana para verificar que se actualice la posición antes de ejecutar el siguiente comando.

III-B. Punto 1c

Para correr este punto se debe usar el siguiente comando (con un mapa ya abierto):
rosrun taller3_4 punto1c.py

Para el Punto 1c aplica el mismo diagrama de flujo con la única diferencia de que se calcula la ruta de Pacman a todas las galletas y se elije la de menor cantidad de movimientos. Al llegar a dicha galleta se realiza el mismo procedimiento para las galletas restantes. El calculo de la ruta a todas la galletas se hizo utilizando el mismo algoritmo de BFS por lo cual solo fue necesario agregar un for y un comparador para lograr este punto.

III-C. Gráfico de nodos

La siguiente imagen muestra el gráfico de los nodos que estaban funcionando (y sus conexiones) cuando se corrieron los programas. Es el mismo Gráfico para ambos puntos (1a y 1c)

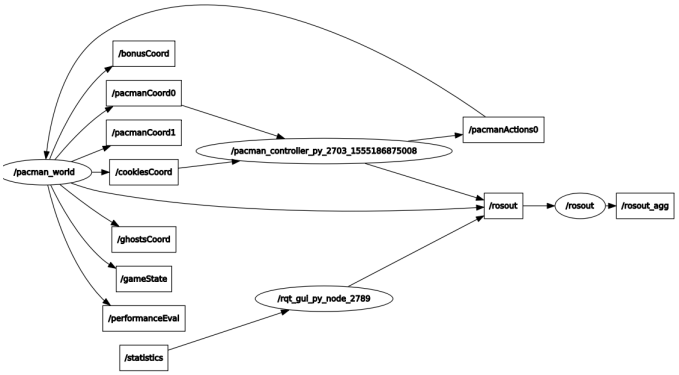


Figura 6: Gráfico punto 1a y 1c

III-D. Análisis de Tiempo

El siguiente análisis se realizo con el algoritmo del punto 1a. Todos los tiempos están en segundos.

Cuadro I: Resultados punto 1.D

Intento	Origen- Galleta 0	Galleta 0- Galleta 1	Galleta 1- Galleta 2	Galleta 2 – Galleta 3	Total Terminar el Mapa
1	0,04249907	0,23455310	0,04478502	0,09575701	32,40054703
2	0,04495096	0,21968508	0,04968715	0,10534310	32,55051804
3	0,02731109	0,21742702	0,03854704	0,06989002	32,85056615
4	0,03580213	0,20929408	0,02585816	0,09288597	31,80050302
5	0,04021406	0,22751403	0,04796290	0,08998203	32,70074701
6	0,01648688	0,22599411	0,04693508	0,09916306	32,40225196
7	0,01682997	0,23362708	0,07512903	0,14838791	32,70053291
8	0,04596591	0,23136187	0,02918696	0,08102298	32,70044613
9	0,04438806	0,24627495	0,03025293	0,09809589	31,95050502
10	0,04664183	0,21002984	0,04455900	0,09758806	32,85048413
11	0,04101610	0,22149897	0,02531886	0,09465885	34,80047894
12	0,01237798	0,23419714	0,08233404	0,09512281	32,70055294
13	0,01721883	0,22269201	0,03616595	0,08657694	31,95063806
14	0,04189897	0,21611190	0,04340291	0,09112000	31,80056000
15	0,04662800	0,22775793	0,04531789	0,09513688	32,85073686
16	0,05252099	0,23143816	0,05094385	0,09775996	32,85074997
17	0,04794097	0,22121286	0,05710411	0,09536409	32,70063901
18	0,04719090	0,21938109	0,04791403	0,09422493	32,55045891
19	0,05094981	0,22809792	0,05776191	0,09422922	32,85054803
20	0,04443812	0,22871900	0,04278612	0,09129310	31,80198693
Promedio	0,03816353	0,22534341	0,04609765	0,09568014	32,58822255
Desv.Estandar	0,012394202	0,00871689	0,01416676	0,014047311	0,63428665

En la tabla de la figura 7 podemos ver el promedio que tomo calcular cada una de las rutas y el total en recolectar todas las galletas. El promedio es mayor para algunas de las rutas, esto depende de la distancia de Pacman a la galleta, ya que entre mas lejos esté el algoritmo deberá expandirse a través de más puntos y más rutas alternas por lo que el cálculo tomará más tiempo para algunas galletas. Si para realizar esta tabla se hubiese usado el algoritmo del punto 1c veríamos un aumento significativo en todos los tiempos, ya que para cada ruta elegida deberá calcular más las rutas a todas las galletas para encontrar la menor,

pero Pacman tomaría menor tiempo en recolectarlas, ya que se iría primero a la más cercana a él.
En cuanto a la desviación estándar, para todos los casos fue bastante baja, ya que en general toma casi el mismo tiempo calcular las distancias. La desviación mas alta fue la de el total para terminar el mapa. Además, se puede ver que este valor si varía mas de 3 segundos en algunas ocasiones, esto podría deberse al tiempo elegido de comunicación entre nodos.

IV. PUNTO 2

IV-A. Punto 2a, 2b, 2c, 2d

IV-A1. Resumen: El literal 2c del taller consistía en crear un nodo en ROS que fuera capaz de llevar el robot Pioneer desde una posición inicial de $(x, y, \theta) = (0, 0, \pi)$ a una posición final pasada por parámetro en consola a la hora de ejecutar el nodo (se deben pasar tres parámetros tipo numero para que se acepte) o usando una posición final por defecto ubicada en $(x, y, \theta) = (39, 39, \frac{\pi}{2})$. Para ello, era necesario que el nodo creado implementara el algoritmo $A*$. En este caso, se empleo una librería para python llamada **networkx** que permite generar grafos y la cual ya posee la implementación de este algoritmo.

IV-A2. Implementación: En primer lugar, se creo una escena en VREP con 5 obstáculos cilíndricos como se especificaba en el primer literal de este punto y los cuales publicaban su posición y diámetro a través de un tópico de ROS. Debido a que el piso de esta escena se asumió como cuadrado con dimensiones de $80x80$ y con una posición $(x, y) = (0, 0)$ se buscaba discretizar el espacio usando el método por descomposición de celdas exactas. Para ello se tomo una medida de discretización del espacio de $0{,}5m$. Por facilidad, también se creo una clase llamada Casilla, la cual representaba un elemento de la discretización del espacio, esta poseía tres atributos, la posición x, la posición y y uno booleano llamado "libre"que era verdadero si no había obstáculo ocupando la casilla y era falso de lo contrario. Por otro lado, se opto por enumerar la discretización del espacio de la siguiente forma que se muestra en la tabla a continuación siendo n^2 el numero de casillas totales de la discretización. Como

0	...	n-1
n	...	2n-1
...	...	...
(n-1)n	...	$n^2 - 1$

Cuadro II: Enumeración de espacio discretizado

se puede ver en la figura de arriba (II) esta enumeración facilito la creación de vértices debido a que se agregaban los elementos tipo casilla un arreglo y su posición en el mismo representaba su representación en el grafo creado importado de la librería netwrokx. Luego se procedió a crear los arcos del grafo, esta tarea se facilito en parte debido al atributo "libre"que se tuvo en cuenta a la hora de generar los elementos tipo casilla. Gracias a esto se iteraba a través de todas las casillas y primero se revisaba si esta estaba ocupada por un obstáculo, de estarlo no se creaban arcos salidos de el, de lo contrario se revisaban sus 8 posibles vecinos, si estos vecinos estaban en los rangos de discretización (en el caso de las casillas de las esquinas o laterales había que considerar que podía tener menos vecinos) y tampoco tenían obstáculos se creaba el arco. Una vez se tenia la representación del espacio en un grafo se uso la implementación de la librería para el algoritmo $A*$ para hallar el camino a la posición final, este método arroja un arreglo con la secuencia de los nombres de los vértices que se deben seguir para llegar a la casilla mas cercana a la posición final pasando como parámetro 3 cosas: el nombre del vértice inicial, el nombre del vértices final y el nombre de cierta función que calculara la heurística entre dos nombres de vértices.Primero, debido a que las posiciones finales e iniciales no necesariamente debían tener las coordenadas exactas de alguna de las casillas desarrolladas se empleo un método auxiliar llamado "numCasillas"que retornaba el numero de la casilla mas cercana para ciertas coordenadas (x, y) pasadas por parámetro de esta forma obtuvieron los primeros dos parámetros necesarios para la implementación de la librería. Por otro lado, se desarrollo un método llamada "heurística"que recibía dos parámetros que representaban los números de los vértices entre los cuales se deseaba hallar la heurística, por simplicidad se empleo la distancia euclidiana entrada posiciones de las casillas, con esto hacemos referencia a que usando tanto el modulo como la división entera del numero de las casillas, era posible obtener la distancia relativa entre ellas sin usar coordenadas exactas.A partir de esto se realizo el siguiente procedimiento, en primer lugar, se creo nuevamente otra estructura de datos llamada Posición que tenia tras parámetros: "x", zz "θ". A partir de esto, se genero un algoritmo que tenia las siguientes variables, primero, había una variable llamada "posicionInter"que representaba la posición a la cual se debía orientar el robot en ese instante del camino, luego una variable ïRuta"que era un índice que marcaba el elemento en el arreglo de datos de la ruta al que se debía orientar el robot, después existía una variable .ªrraivedP"booleana que a la hora de actualizar las velocidades de los motores verificaba si el robot estaba lo suficientemente cerca de su posición de destino para poner $\rho = 0$ y $\alpha = 0$ y de estar en medio de la ruta orientarse al siguiente elemento de la ruta (también podía a la posición final) o de haber llegado a la posición final, enfocarse en corregir el θ final deseado, como se puede ver en el algoritmo el ángulo entre el eje de referencia global y local no se tiene en cuenta en la mayoría de casos mas halla de buscar orientar el camino ligeramente y solo se tiene fuertemente en cuenta cuando la posición del Pioneer esta en cercanías (0.1m) a la posición final para corregir el ángulo final deseado. Por otro lado, se empleo otro nodo graficador que actualizaba el recorrido del Pioneer, esto nodo solo tenia dos suscripciones a los tópicos de los obstáculos y de la posición del robot y usando la librería matplotpy (el modulo animate) se iban actualizando dos vectores de coordenadas del Pioneer, este nodo se ejecutaba usando ROSCORE desde el nodo antes descrito por lo que solo es necesario ejecutar uno de ellos.

IV-B. Grafo de ROS

A continuación se muestra la imagen con el grafo de ROS para la solución planteada.

Figura 7: Grafo de ROS Punto 2c

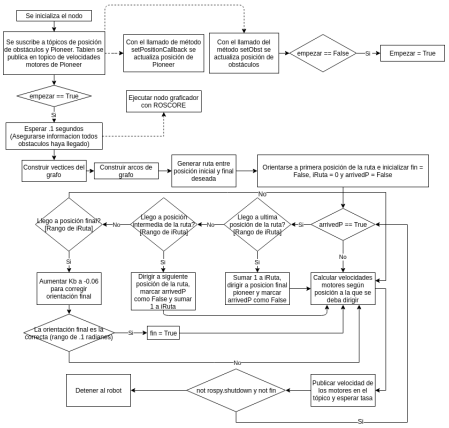


Figura 8: Diagrama de flujo punto 2c

Como se ve en la imagen 7 la solución planteada estaba compuesta por dos nodos, sin contar el nodo de VREP. Uno que implementaba toda la lógica computacional para la discretizacion del espacio y control del robot y otro para la generación del gráfico. Por otro lado, el grafo esta compuesto por tres tópicos. El primero de los tópicos hace referencia al llamado ÏnfoObs.en el cual el nodo de VREP publicaba la posición y dimensiones de los 5 obstáculos cilíndricos usando mensajes tipo Twist donde las coordenadas lineales "x z representaban su localización y las coordenada "z.el diámetro del cilindro, por otro lado, en las coordenadas angulares solo se empleo la "x"debido a que en ella se publicaba el numero con el que se identificaba al obstáculo. A este tópico se suscribían tanto el nodo graficador como el del punto 2c con el fin de que se pudiera tanto graficar los obstáculos como generar correctamente los vértices y arcos del grafo de discretizacion del espacio. Por otro lado, el segundo tópico hacia referencia al llamado "pioneerPosition por el cual el nodo de VREP publicaba la posición, su funcionamiento era similar al del taller pasado por lo que no se profundizara en su funcionamiento, tan solo se dirá que nuevamente el graficador y el nodo de la solución se suscribían para poder obtener esta información. Por ultimo, el tópico "motorsVel.era el tópico por el cual se comunicaba la velocidad que debían poseer los motores del Pioneer, en este tópico se ejercía de forma directa la ley de control del nodo del punto y por eso este publicaba usando un arreglo de valores las velocidades de cada uno de los motores. Por otro lado, el único nodo suscrito a este tópico era el de VREP para actualizar los valores de la simulación.

IV-B1. Descripción de alto nivel : También, se muestra a continuación un diagrama de flujo que detalla la solución de alto nivel para el punto 2c, en ella se explica el procedimiento tomando para orientar el robot hasta el punto final (esta imagen se puede detallar mejor en la sección de anexos).

IV-C. Punto 2e, 2f, 2g

IV-C1. Resumen: En este literal del taller se buscaba llevar al robot Pioneer de $(x, y, \theta) = (0, 0, \pi)$ a una posición final pasada por parámetro en consola a la hora de ejecutar el nodo (se deben pasar tres parámetros tipo numero para que se acepte) o usando una posición final por defecto ubicada en $(x, y, \theta) = (39, 39, \frac{\pi}{2})$. Sin embargo, en este caso se debía implementar el algorítmo de Rapidly Exploring Random Trees (RRT). Para crear las conexiones entre nodos se utilizó la librería networkx, pero, la implementación del algoritmo RRT fue implementada por nosotros.

IV-C2. Implementación: Para estos puntos se utiliza la misma escena creada en el literal anterior. Para este punto se buscaba que el árbol se fuera expandiendo de manera aleatoria hacia coordenadas diversas del mapa hasta que se encontrara una ruta hasta la posición final delimitada. Para ello, se utilizan dos generadores de números aleatorios que definen coordenadas en el espacio por cada iteración. A partir de un método se determina la casilla más cercana al recorrer todos los nodos creados hasta el momento y retornar las coordenadas de esta casilla mas cercana al punto aleatorio creado. A partir de esta casilla mas cercana y un "step.o paso delimitado entre cuadrículas se calcula la coordenada x e y (a partir del teorema de Pitágoras) en la que podría haber una nueva casilla de acuerdo con lo delimitado. Se procede a verificar que la distancia entre el nuevo nodo propuesto y el centro del obstáculo más cercano sea superior al radio del obstáculo y la diagonal del robot Pioneer (esto último dado que los nodos se están modelando como puntos y aún no toman en cuenta las dimensiones del robot). Si lo anterior se cumple se adhiere el nuevo nodo al grafo y se establece una conexión con la casilla más cercana, de lo contrario, no se adhiere. Todo lo anterior puede quedar representado con mayor claridad en el siguiente diagrama:

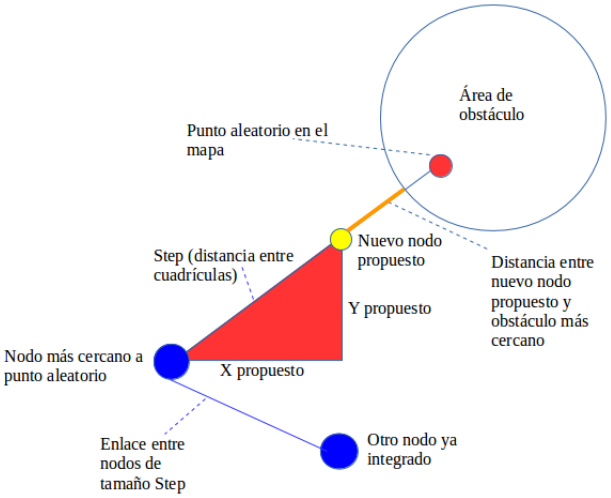


Figura 9: Diagrama de descripción del método RRT

Cabe recalcar que durante este proceso un arreglo ”track.ªlmacena las posiciones de las cuales cada nodo provino para posteriormente poder recuperar la ruta. El algoritmo termina cuando se crea una casilla los suficientemente cercana (dado un parametro pre delimitado) a la posición a la que se desea llegar. A partir de un método llamado ”trackRoutez el arreglo ”track”se recupera la ruta hasta el punto final y se procede al movimiento del robot hasta este punto igual a como se comento en el literal anterior.

IV-C3. Descripción de alto nivel : También, se muestra a continuación un diagrama de flujo que detalla la solución de alto nivel para el punto 2e, en ella se explica el algoritmo RRT implementado, cabe recalcar que dado que en el diagrama del punto 2c ya se explica como se llega a la posición final, en este diagrama este proceso no se muestra al igual que otras explicaciones ya dadas en el punto 2c.

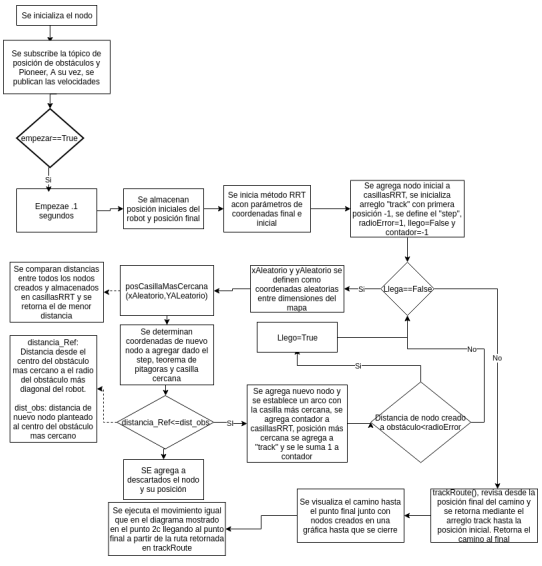


Figura 10: Diagrama de flujo punto 2e

Este diagrama también se detalla mejor en anexos.

IV-C4. Gráfico de nodos: En la siguiente gráfica se muestran los nodos y sus conexiones cuando se corre el programa. Esto específicamente para el punto 2e:

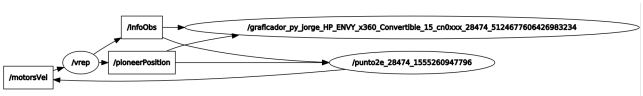


Figura 11: Grafo de punto 2e

Como se puede evidenciar en la figura anterior, los tópicos /InfoObs y /pioneerPosition se comunican la infromación del nodo0 /vrep a los nodos /graficador y /punto2e. El graficador permite mostrar la gráfica en tiempo real y el nodo /punto2e permite, a través del tópico /motorsVel, comunicarse con el nodo /vrep y mover así el robot Pioneer.

IV-C5. Análisis de Tiempo: Se realizó una tabla ejecutando el programa creado en el punto 2e 20 veces, calculando cada vez el tiempo en que toma calcular la ruta, la longitud de la ruta encontrada y el número de nodos creados. LO anterior, partiendo de la posición $(x, y, \theta) = (0, 0, \pi)$ y llegando a la posición final de $(x, y, \theta) = (10, 10, 0)$. Además se calcula la desviación estándar y el promedio de los datos tomados.

Cuadro III: Resultados punto 2.g partiendo de posición inicial $(x, y, \theta) = (0, 0, \pi)$ y llegando a posición final $(x, y, \theta) = (10, 10, 0)$.

Ejecución	Tiempo en calcular la ruta (segundos)	Distancia hasta punto final (metros)	Número de nodos creados (#)
1	8,01	65,26	5136
2	11,33	44,20	6386
3	0,05	42,83	331
4	0,53	40,29	1494
5	3,62	57,61	3734
6	16,56	96,23	7630
7	3,63	68,91	3712
8	0,85	66,77	1821
9	0,01	35,16	95
10	0,04	37,06	144
11	0,09	40,40	575
12	0,25	37,80	1051
13	0,57	51,65	1594
14	2,78	98,39	3269
15	2,13	47,98	2906
16	0,03	47,44	145
17	0,75	39,38	1737
18	1,18	42,84	2205
19	0,93	40,54	2034
20	5,19	38,95	4434
Promedio	2,93	51,98	2521,65
Desv. Estandar	4,38	18,53	2132,65

Como se evidencia en la tabla anterior, la distancia, el tiempo y el número de nodos creados son proporcionales entre si. A medida que hay más nodos, el algoritmo tarda un mayor tiempo en calcular la ruta y a su vez, aumenta la distancia hasta el punto final., El promedio que se tarda en calcular la ruta es de 2,93 segundos con una desviación estándar relativamente baja. Hay una gran desviación estándar entre los números de nodos creados y la distancia hasta el punto final debido a la aleatoriedad del método. Por esta aleatoriedad, las distancias entre el punto final y el inicial se pueden acortar dependiendo de hacia donde empiece a crecer el árbol creado.

V. PUNTO 3

En este tercer y último punto del taller se debía representar la información entregada por un sensor láser. Para esto se realizó una escena la cual se encuentra dentro del archivo *resources* en donde se agregó al robot junto con su sensor láser y 5 obstáculos cúbicos de diferentes tamaños y orientados de manera variada. El script en Lua asociado al sensor láser es aquel entregado junto al enunciado del taller y es el que se utilizó para resolver tanto el iteral a como el c. En el iteral a únicamente se requería mostrar los puntos crudos entregados por el láser, mientras que en el c se debían graficar las líneas obtenidos luego de utilizar el algoritmo de RANSAC junto con el ajuste de línea mediante el método de mínimos cuadrados. El procedimiento de cada iteral se detallarán a continuación en donde también se muestran los diagramas de flujo y grafos correspondientes.

V-A. Punto 3a

Para resolver este punto era necesario subscribirse al tópico *scanner*. En este tópico se encuentra la información detectada por el sensor láser. El formato de dicho tópico consiste de un ángulo ρ y una distancia d. El ángulo ρ hace referencia a los radianes en donde se tomó cada muestra (con respecto al eje z) y la distancia d se refiere a la distancia entre el punto detectado y el sensor. Es muy importante notar que estas medidas son con respecto al plano local del robot, por lo cual cuando sea necesario graficarlas en una posición global se debe realizar su debida transformación al plano global de referencia. Para convertir las coordenadas al plano global basta con sumar las coordenadas del robot. Se desarrolló un script en Python para resolver el problema del enunciado. En este se inicializa un nodo el cual se encarga de mover al robot y otro nodo adicional que se encarga de graficar la posición actual del robot y los puntos detectados por el sensor láser. La tasa a la cual funciona el primer nodo se definió como la misma del tópico scanner. Esta tasa se encontró utilizando el comando *rostopic hz scanner* y es de 18Hz. Para ejecutar los scripts e inicializar los nodos se debe correr el siguiente comando:

rosrun taller3_4 punto3a.py

En la siguiente Figura se puede ver el resultado de ejecutar el comando *rqt_graph* en la ventana de comandos al realizar la instrucción anterior.

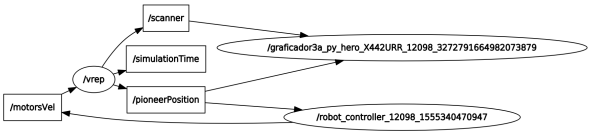


Figura 12: Grafo para el punto 3a

En la Figura 12 se pueden ver las conexiones entre los nodos. Existe un tópico *pioneerPosition* en donde se publica la posicion del robot, otro topico *scanner* en donde se encuentran los puntos crudos detectados por el sensor y además otro topico *motorsVel* en el cual se publica la velocidad de cada motor de acuerdo con las teclas presionadas. Se tienen 2 nodos: *robot_controller* que se encarga de detectar las teclas presionadas y publicar las velocidades correspondientes en el topico *motorsVel* y el nodo graficador *graficador3a* el cual se encarga de graficar la posicion actual del robot y los puntos crudos detectados por el sensor. El procedimiento que realiza cada nodo se puede ver en el siguiente diagrama de flujo.

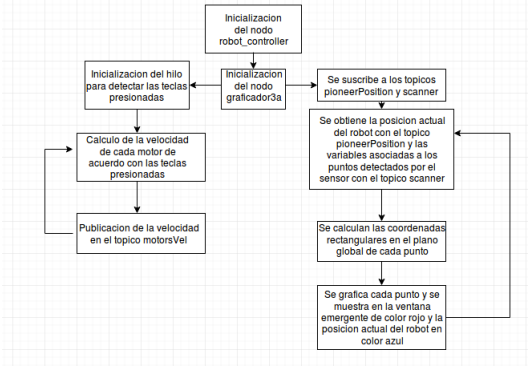


Figura 13: Diagrama de Flujo Punto 3a

V-B. Punto 3c

En el iteral c del Punto 3 se debían extraer características de los puntos detectados por el sensor. El objetivo era utilizar el algoritmo RANSAC junto con un ajuste de líneas con el método de mínimos cuadrados para pintar las lineas obtenidas de los puntos crudos. Este proceso se realizó mediante 2 scripts en Python, uno se encarga de obtener los puntos crudos del tópico *scanner* y calcular las líneas, mientras que el otro se encarga únicamente de graficar la posición actual y las líneas resultantes. Este relación se puede ver de manera más clara en el siguiente grafo resultado de ejecutar el comando *rqt_graph*. La tasa se escogió también como 18Hz por las mismas razones que el iteral anterior.

Figura 14: Grafo para el punto 3c

En la Figura 14 se puede ver un tópico adicional al iteral anterior llamado *rectas*. En este tópico se publica la información requerida para graficar cada recta, trabajo realizado por el nodo graficador *graficador3c*. El formato de dicho tópico utiliza un mensaje de tipo *Float32MultiArray* en donde se guardan 4 variables distintas para cada rectas: la pendiente, el intercepto en el eje Y (también llamado el offset) y los valores en x mínimo y máximo. Los valores en x se utilizan para conocer el intervalo en donde se pinta la línea y las otras 2 variables son suficientes para construir la ecuación de la recta y calcular los valores en y equivalentes al intervalo en x. Estas 4 variables equivalen a 1 recta, por lo cual el tamaño del arreglo de rectas varía

dependiendo de cuantas rectas se calculen equivaliendo a 4 veces el número de rectas. El procedimiento realizado en cada script de Python se puede detallar en el siguiente diagrama de Flujo.

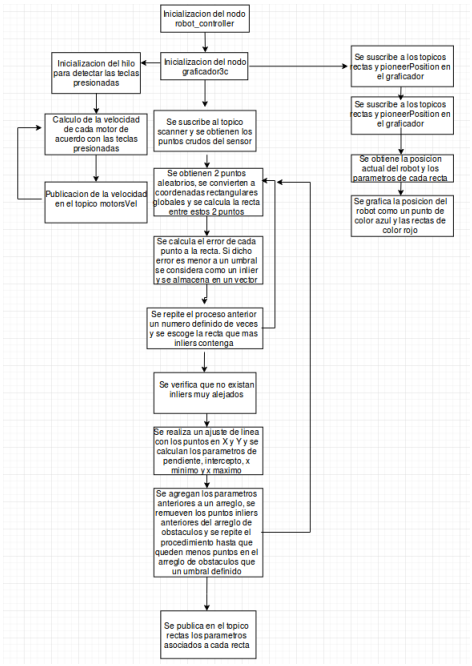


Figura 15: Diagrama de Flujo Punto 3c

VI. Anexos

VI-A. Punto 1

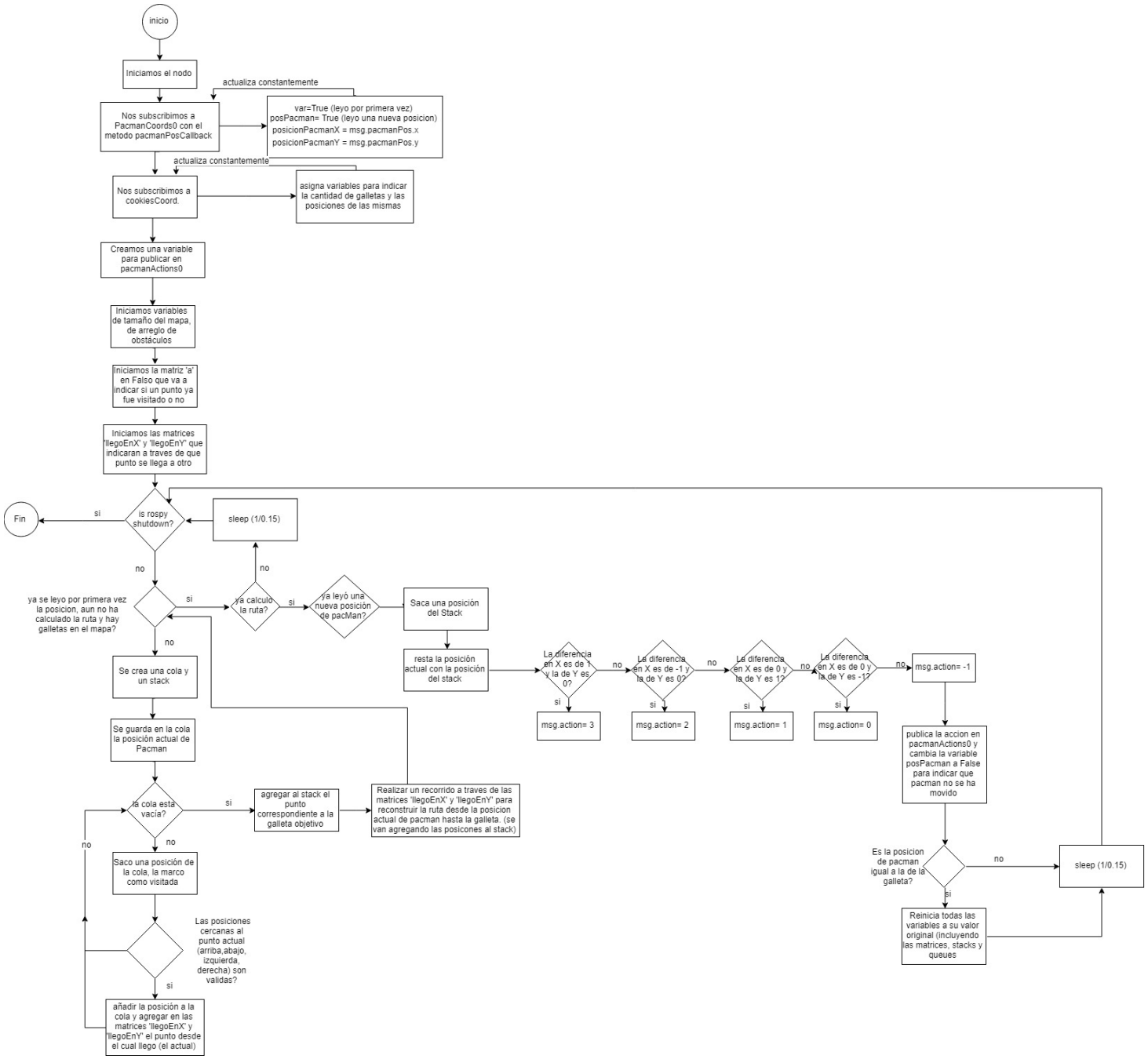


Figura 16: Diagrama punto 1

VI-B. Punto 2

VI-C. Punto 3

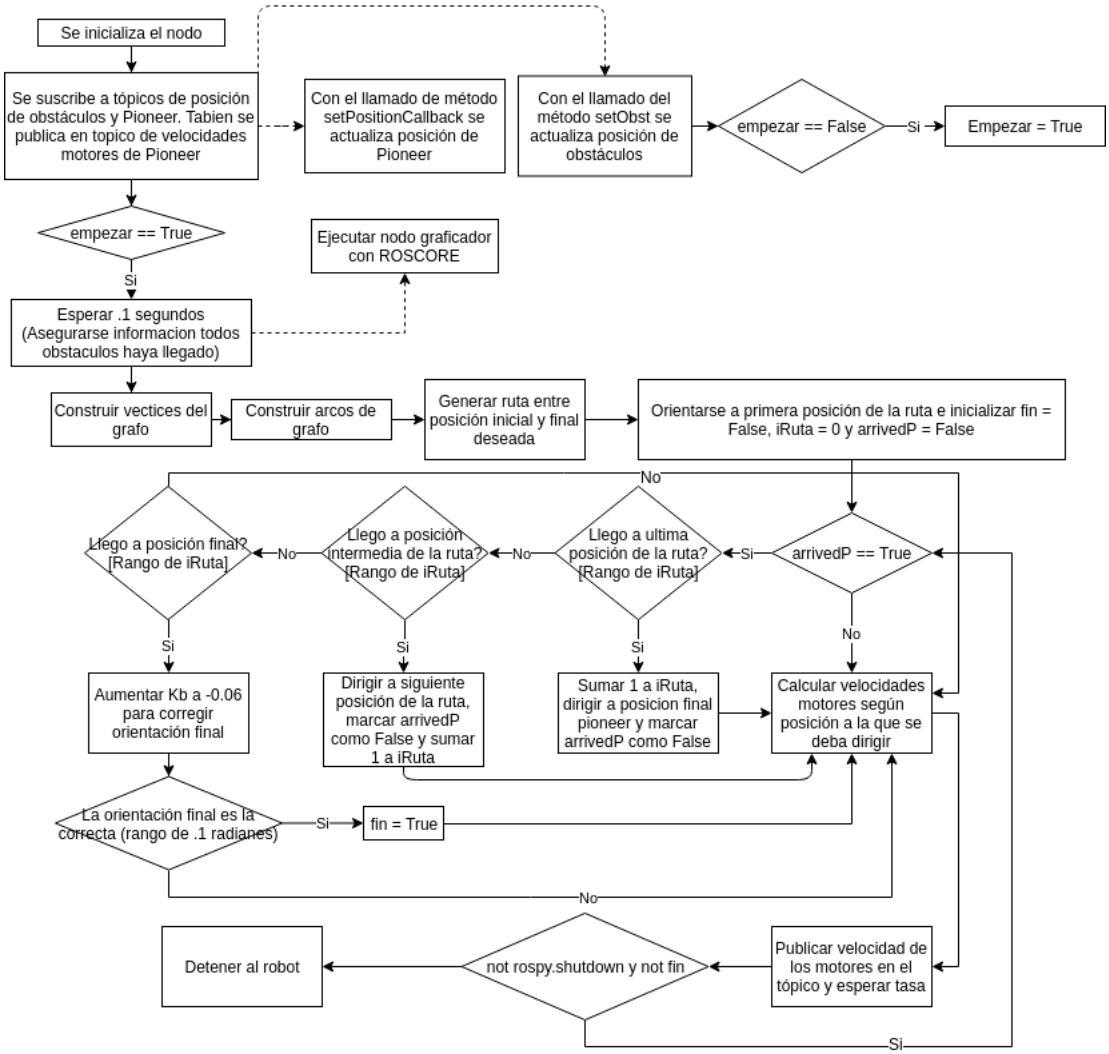


Figura 17: Diagrama de flujo punto 2c

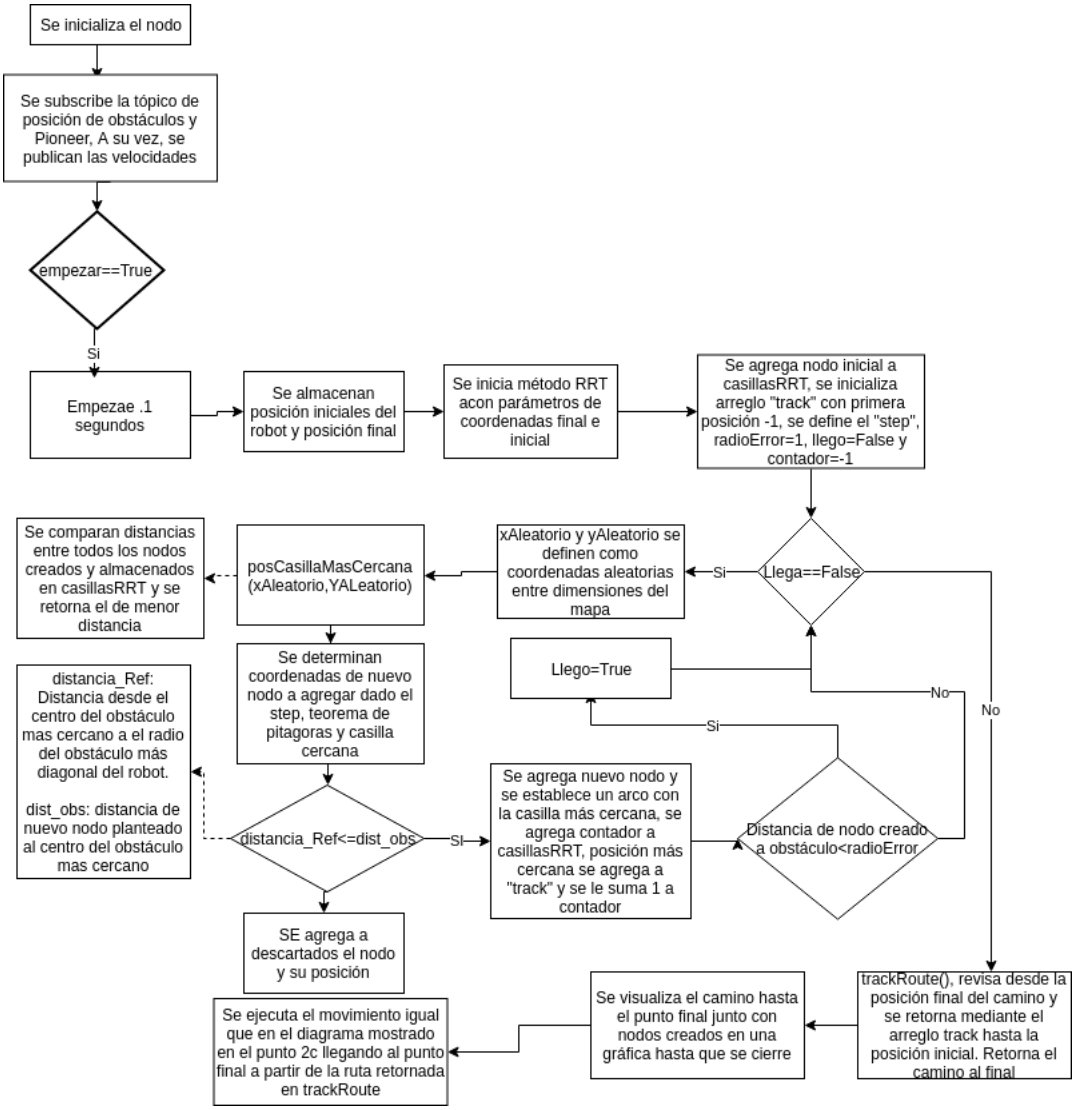


Figura 18: Diagrama de flujo punto 2e

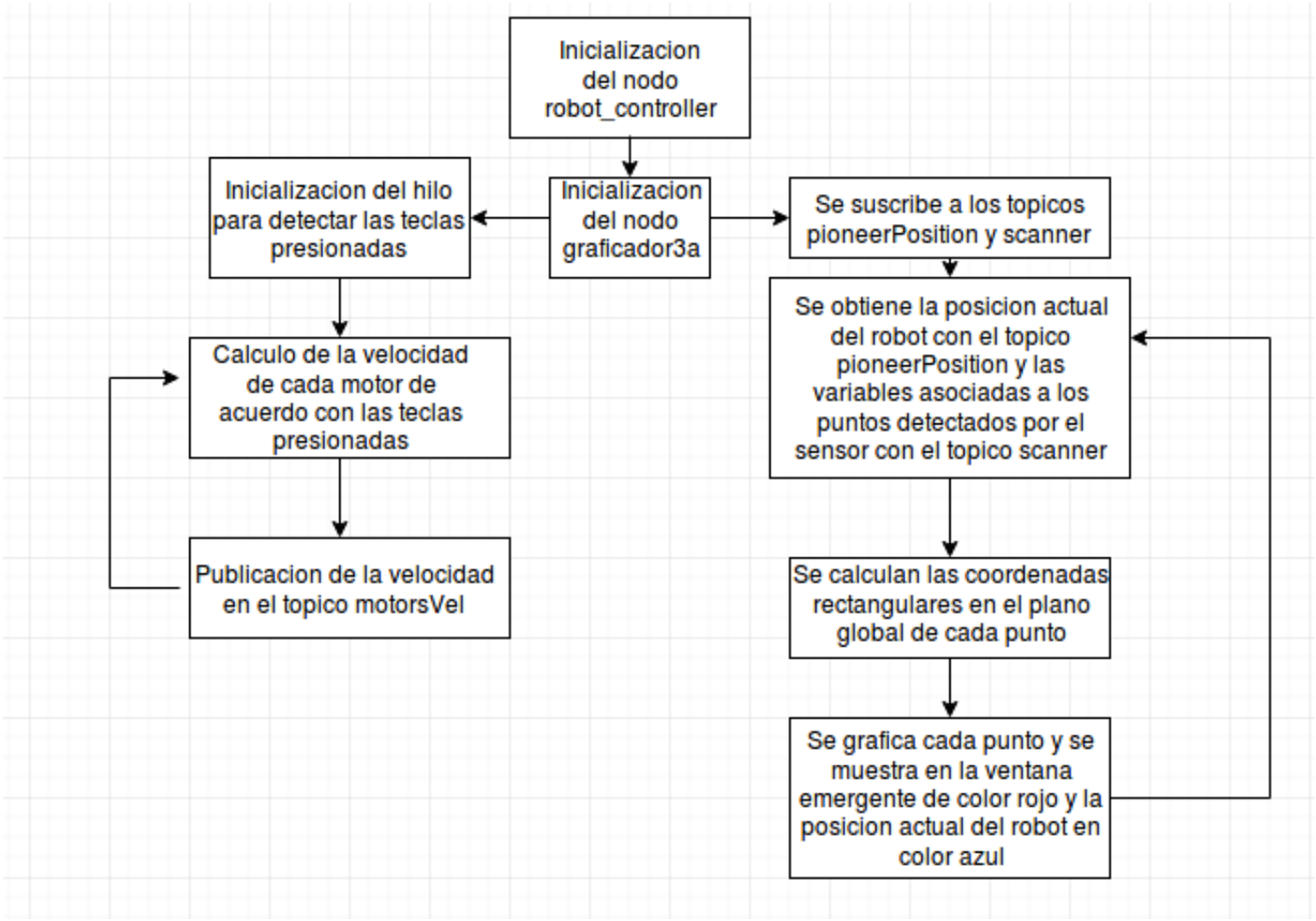


Figura 19: Diagrama de Flujo punto 3a

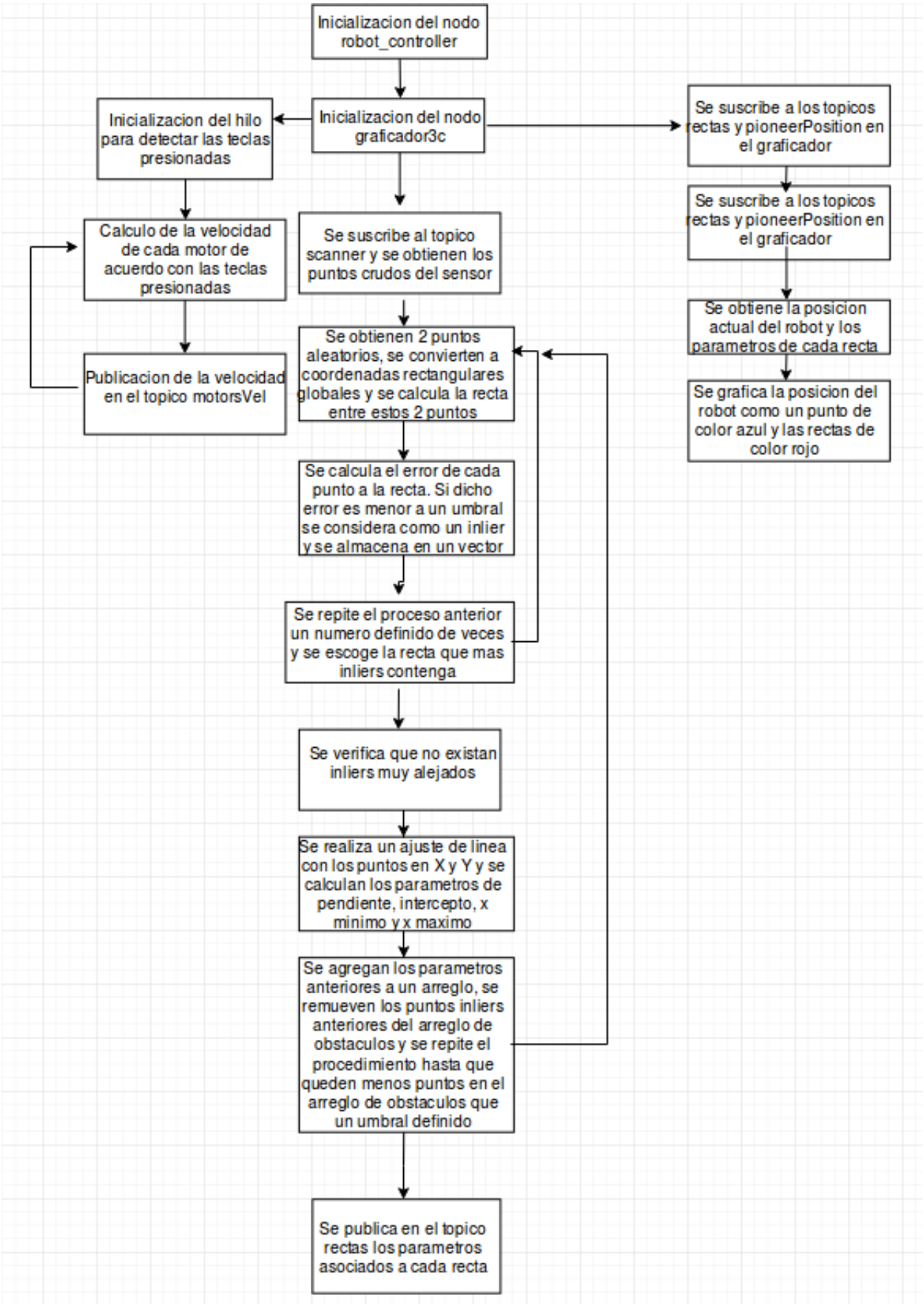


Figura 20: Diagrama de Flujo punto 3c