Universidad Zaragoza
1542

Trabajo Fin de Grado

Implementación de algoritmo de control Pure Pursuit en robots móviles ARDUINO y comparación con otros algoritmos existentes

Implementation of Pure Pursuit control algorithm in ARDUINO mobile robots and comparison with other existing algorithms

Autor

Diego Sangüesa Pérez

Director

Cristian Mahulea

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2017

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Diego Sangüesa Pérez,

con nº de DNI 25206868-H en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Implementación de algoritmo de control Pure Pursuit en robots móviles ARDUINO y comparación con otros algoritmos existentes.,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 19 de Septiembre de 2017

Fdo: Diego Sangüesa Pérez

AGRADECIMIENTOS

Agradezco a la Universidad de Zaragoza, y más especialmente a la EINA, por la oportunidad de llevar a cabo este proyecto en sus instalaciones, así como por todos los medios prestados para tal fin. También quiero agradecer enormemente tanto a mi director Cristian Mahulea, por su guía, sus consejos, su buena disposición para ayudarme con cualquier problema y su gran rapidez al atenderme en cualquier momento; como a Emanuele Vitolo, por el tiempo dedicado, por compartir conmigo sus conocimientos y sus proyectos, y por todos los cafés que nos hemos tomado juntos; sin ellos no habría sido capaz de hacer este trabajo. Por último, pero no por ello menos importante, quiero agradecer a todos mis amigos y compañeros por su apoyo y sus consejos, siempre dispuestos a animarme, así como a toda mi familia, especialmente a mis padres, por aguantarme y entenderme en los malos momentos, y estar siempre ahí con su comprensión y su cariño.

Implementación de algoritmo de control Pure Pursuit en robots móviles ARDUINO y comparación con otros algoritmos existentes

RESUMEN

La integración de robots móviles en diversos ámbitos, como el industrial o el doméstico, es de gran importancia, ya que éstos facilitan la realización de ciertas tareas para las cuales en otro caso sería necesaria la presencia de una persona. Además, esto puede reportar importantes beneficios como un ahorro de tiempo, de personal o de esfuerzo, o incluso mejoras en la precisión, el rendimiento o la productividad. Este tipo de robots poseen un método de control que les permite seguir una trayectoria calculada a partir de la información recibida por diversos sensores. Uno de los métodos de control más utilizados, debido a su simplicidad y a los buenos resultados que aporta es el Pure Pursuit, que además sirve de base para el desarrollo de métodos más complejos.

Este trabajo consiste en la implementación de un controlador del tipo pure pursuit en robots móviles Arduino. Para ello se ha trabajado con una plataforma ubicada en uno de los laboratorios del edificio Ada Byron que fue desarrollada por otro estudiante en su Trabajo de Fin de Máster. En primer lugar se ha estudiado el funcionamiento de los controles del tipo Pure Pursuit, y posteriormente se ha decidido la mejor forma de adaptar un controlador de este tipo a los robots de la plataforma citada anteriormente.

Se ha escrito y depurado el código del algoritmo, eliminando todos los fallos y defectos aparecidos y adaptándolo para su adecuado funcionamiento junto con las necesidades de la plataforma. Una vez escrito el código, se han ajustado los parámetros del control a los valores más adecuados mediante experimentación, hasta obtener un comportamiento del robot lo más próximo posible al deseado.

Asimismo, se ha utilizado el software Matlab para la simulación del comportamiento del robot en diferentes escenarios, variando diferentes parámetros de funcionamiento y simulando también dichos escenarios con un control del tipo PI a fin de comparar ambos métodos.

Índice

Capítulo 1

Introducción y objetivos

El objetivo principal de este proyecto es el estudio de los métodos de control de robots del tipo Pure Pursuit y su adaptación a robots Arduino para su posterior uso en la plataforma ubicada en el edificio Ada Byron.

Además, se pretende realizar un estudio de diversos parámetros importantes en el control y el funcionamiento y elegir los más adecuados para el robot Arduino que se va a utilizar en la plataforma.

Para ello, las herramientas que vamos a utilizar principalmente son:

– La plataforma de robots:

  Diseñada y desarrollada por otro estudiante para su Trabajo de Fin de Máster (Emanuelle Vitolo), consta de una base que se extiende sobre el suelo, diferentes códigos identificadores pegados en cartulinas y una cámara suspendida sobre la base, colocada en un andamio y conectada al ordenador. De esta manera, al iniciar el programa que rige el funcionamiento de la plataforma, la cámara transmite la imagen al ordenador, el cual debe reconocer los indicadores anteriormente mencionados colocados en las cuatro esquinas de la base, otro indicador colocado sobre el robot en su posición inicial y con la orientación adecuada, y la posición de los obstáculos que hayamos deseado colocar. Dichos obstáculos son simplemente figuras de cartulina con un color diferente al de la base y colocadas en la posición deseada sobre ésta[1].

  El ordenador pide un punto de destino, el cual debemos introducir mediante las coordenadas de la base, y hecho esto calcula una ruta desde el punto inicial en el que se encuentra el robot hasta el punto de destino y evitando pasar por los

obstáculos. Para hacer esto, el ordenador divide la superficie de la base en una rejilla de cuadrados de 25 por 25 centímetros, y selecciona una serie de centros de estos cuadrados por los que el robot debe pasar para llegar al destino sin colisionar con los obstáculos. De esta manera, la trayectoria que debe seguir el robot es una serie de lineas rectas, que unen cada uno de los puntos principales que conforman la trayectoria.

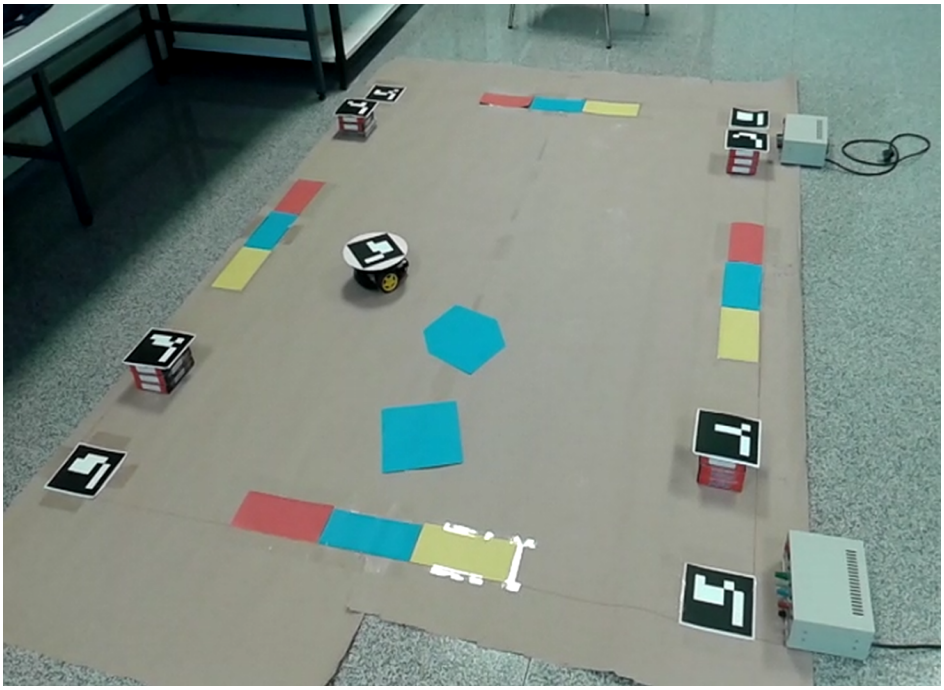

Figura 1.1: Plataforma de robots móviles

Para comunicarse con el robot, se conecta al ordenador un periférico capaz de enviar paquetes de datos mediante señales wifi. La plataforma tiene la capacidad de comunicarse con el robot en tiempo real, comprobando la posición en la que se encuentra el robot mediante la cámara en cada momento, pero para este trabajo se utilizará un estimador interno en el código del robot, de manera que el ordenador solo mandará un paquete wifi inicial al robot en el que aparecerá la posición y la orientación inicial de éste y los puntos de la trayectoria que debe recorrer, y el robot se moverá de manera autónoma calculando su posición a partir de la velocidad que transmita a sus ruedas.

– Un robot Arduino:

El controlador que se va a desarrollar será implementado en un robot móvil ARDUINO. Este tipo de robot tiene una serie de características que es

necesario tener en cuenta a la hora de diseñar un algoritmo de control para él. En primer lugar, hay que ser consciente de qué significa que sea un robot móvil. Se considera robot móvil a aquel robot que no está situado en una base fija ni conectado de forma permanente a un sistema, sino que se puede mover libre e independientemente. Debe contar con un sistema de baterías para almacenar energía, así como una memoria y un procesador en los que guardar y ejecutar el código con el algoritmo deseado. Con todo esto, el robot funcionará de manera autónoma hasta que se acabe su batería y sea necesario recargarlo.

Para permitir su movimiento, el robot cuenta con dos ruedas, una a cada lado, y una esfera en la parte frontal que sirve de tercer punto de apoyo y gira en cualquier dirección sin oponer resistencia, actuando como rueda loca. Este robot usa un mecanismo de dirección del tipo "differential drive", lo cual quiere decir que a pesar de que sus dos ruedas comparten el eje de rotación, no están unidas por un eje físico como las de los automóviles, sino que cada una de ellas cuenta con un motor independiente que les permite girar a distinta velocidad o incluso en distinto sentido [2].

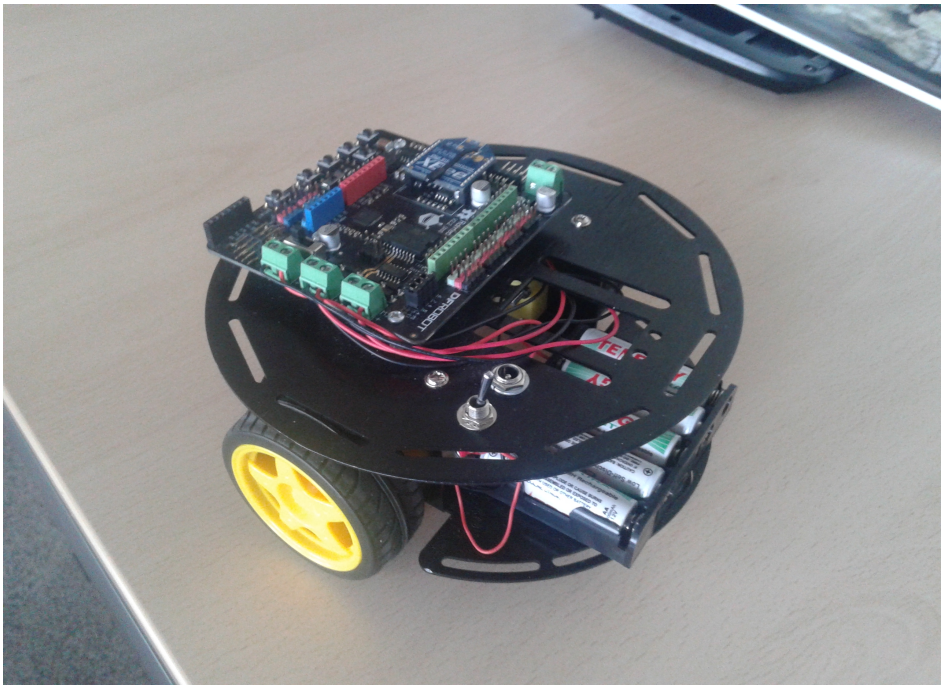

Figura 1.2: Robot utilizado

De esta manera, si ambas ruedas giran en el mismo sentido y a la misma velocidad el robot se desplazará siguiendo una trayectoria recta; si giran a la misma velocidad pero en sentidos distintos, el centro de giro se ubicará en el

punto medio de la linea que une ambas ruedas, lo cual coincide aproximadamente con el centro del robot, es decir, el robot girará sobre sí mismo; y si giran en el mismo sentido pero a distintas velocidades el centro de giro se encontrará en algún punto del eje que une ambas ruedas pero no en el tramo que se encuentra entre ambas.

Este sistema de dirección es muy comúnmente usado en la robótica, ya que su control es muy sencillo, pero tiene la desventaja de ser muy sensible, dado que cualquier pequeña diferencia en la velocidad relativa entre las ruedas o incluso irregularidades en el terreno pueden variar significativamente la trayectoria. Por este motivo, es importante ajustar bien las velocidades, ya que ambos motores pueden tener un comportamiento ligeramente diferente a pesar de ser iguales, lo que perjudicaría la precisión del seguimiento de la trayectoria.

Por último, cabe mencionar que ARDUINO utiliza el lenguaje C++, de manera que este deberá ser el que se utilice para escribir el código del controlador.

– Matlab:

Se trata de un software ampliamente usado debido a su versatilidad, ya que nos permite desde hacer cálculos sencillos a gráficas o simulaciones de alta complejidad, así como desarrollar un código o algoritmo. Matlab es capaz de trabajar con grandes cantidades de datos y realizar análisis a partir de ellos, puede transformar código creado en el propio Matlab a otros lenguajes de programación para que su implementación en diferentes hardware sea más rápida y sencilla, y gracias a Simulink posee un entorno en el que se puede trabajar con diagramas de bloques y modelos gráficos que facilitan en muchas ocasiones el desarrollo de algoritmos. Por todos estos motivos, se trata de un software usado mundialmente, tanto a nivel de estudiantes como a nivel profesional, en muy diversas aplicaciones [3].

En este caso particular, será utilizada una “toolbox” de robótica (http://webdiis.unizar.es/RMTool/) para simular el comportamiento de un robot parecido al robot ARDUINO para el que se desarrollará el controlador en la realidad. Al ejecutar dicha toolbox se abre un entorno en el que es posible diseñar gráficamente los obstáculos con los que se va a encontrar el robot, así como fijar los puntos de partida y de destino. También se permite cambiar

diversos parámetros del robot o su modo de funcionamiento, tales como su velocidad tanto en linea recta como de giro, el diámetro de sus ruedas, la distancia de "lookahead", el controlador que rige el comportamiento del robot, el modo de calcular la trayectoria, el tipo de robot entre differential drive o un modelo similar a un automóvil...

A partir de esta toolbox se harán diferentes simulaciones de posibles situaciones en las que se encontrará el robot ARDUINO. Así, con unos mismos obstáculos y puntos de partida y destino, se simulará el comportamiento del robot tanto con un controlador del tipo Proporcional Integral (PI) como con un controlador Pure Pursuit, y también se variarán parámetros como la velocidad de avance y de giro del robot o la distancia de "lookahead", para así poder comparar los distintos resultados y, atendiendo a datos obtenidos de la simulación como el tiempo transcurrido o el error medio respecto a la trayectoria original, decidir qué valores son los más adecuados para el comportamiento óptimo del robot y que ventajas o desventajas presentan.

1.1. Planteamiento del trabajo

En primer lugar se ha de estudiar y comprender en qué consiste un controlador Pure Pursuit, como funciona, que parámetros importantes lo definen, que limitaciones tiene, etc. Una vez hecho esto, hay que comprender el funcionamiento de la plataforma de robots, ya que se trata del sistema donde queremos implementar el controlador, y es importante saber que datos es capaz de enviarle al robot, que tipo de situaciones podemos encontrarnos o como debe responder el robot.

A continuación, hay que pensar cómo adaptar un algoritmo de control Pure Pursuit a nuestro robot. Conociendo el tipo de trayectorias que deberá recorrer el robot y cómo se comporta, se debe adaptar el principio de funcionamiento de un algoritmo Pure Pursuit para que el robot llegue al destino de manera deseada, en un tiempo razonable, evitando los obstáculos y sin hacer más cálculos de los necesarios. Además, dado que se debe programar el controlador en lenguaje C++ para que funcione en el robot ARDUINO, hay que tener un conocimiento base de este lenguaje de programación, por lo que es necesario repasar las instrucciones y sintaxis básicas.

Cuando se tiene el código y se ha depurado y eliminado los errores, el siguiente paso es la experimentación con el robot y la plataforma. En esta fase, viendo los

fallos que puede cometer el robot una vez que posee el controlador final, es necesario ajustar los parámetros propios del movimiento del robot como la velocidad de giro, la de avance o la distancia de "lookahead", a fin de ajustar el comportamiento hasta conseguir el óptimo.

Por otro lado, gracias a la ayuda de Matlab y la toolbox de movimiento de robots desarrollada en el Área de Ingeniería de Sistemas y Automática de la Universidad de Zaragoza, se deben hacer diversos experimentos simulando la situación del robot real en la plataforma, y utilizando diversos controladores y valores para los parámetros representativos, a fin de conocer como y en qué medida afecta cada parámetro, cuales son los más sensibles, que problemas se pueden encontrar a la hora de implementar un control parecido en otros hardware, y facilitar la comprensión de su funcionamiento.

Capítulo 2

Algoritmo Pure Pursuit

2.1. Funcionamiento general

Los algoritmos de control del tipo Pure Pursuit, o en español *persecución pura*, son muy usados en aplicaciones de robots móviles, ya que suelen ser sencillos de implementar y dan buenos resultados, el seguimiento de la trayectoria es bastante preciso y el tiempo hasta llegar al destino no es elevado.

Como su propio nombre indica, la idea principal de este tipo de algoritmos es que el robot persigue un punto al que nunca llega. Para utilizar un controlador de este tipo es necesario que se conozca la trayectoria que debe seguir el robot de antemano, o por lo menos los siguientes puntos por lo que debe pasar. Una vez conocida la trayectoria, el controlador busca en ella un punto que se encuentre a una determinada distancia de la posición actual del robot, que debe ser conocida en todo momento.

Esta distancia es lo que se conoce como distancia de "lookahead", ya que el robot "mira hacia delante" en la trayectoria para buscar el punto, y es el parámetro más importante a ajustar en este tipo de controladores, por lo que dependiendo de robot y del entorno en el que se vaya a mover se elegirá una distancia u otra. Además, no existe una fórmula o un valor ideal para elegirla, depende enteramente de las circunstancias, e incluso es posible que para un mismo robot la distancia de "lookahead" que proporciona un mejor comportamiento varíe al cambiar la trayectoria que éste debe seguir.

Como ya se ha dicho, para utilizar un controlador de este tipo, el robot debe conocer tanto la trayectoria que va a seguir como su posición en cada instante para poder encontrar el punto objetivo que pertenezca a la trayectoria y se encuentre a la distancia de "lookahead" de su posición actual. Una vez encontrado el punto objetivo,

es tratado como el destino del robot, por lo que el robot debe acercarse a él, cosa que continua haciendo hasta que en uno de los ciclos detecta que la distancia que lo separa de ese punto objetivo es menor que la distancia de "lookahead". En ese momento, se vuelve a buscar un punto objetivo más lejano, a una distancia igual o mayor que la de "lookahead", y este será considerado el nuevo punto objetivo.

Con este procedimiento, el robot va avanzando por la trayectoria sin detenerse, siguiéndola más o menos estrechamente en función de los parámetros fijados, sobretodo de la distancia de "lookahead", hasta que llega el final de la trayectoria. Si la distancia de "lookahead" que se ha fijado es elevada, se evitará que el robot haga oscilaciones innecesarias alrededor de la trayectoria después de un giro, pero si es demasiado elevada tomará el giro de la trayectoria mucho más cerrado, de manera que si el giro se hacía para evitar un obstáculo y no había mucho margen es posible que llegue a colisionar. En cambio, si se elige una distancia de "lookahead" muy baja, el robot seguirá la trayectoria de forma mucho más estricta, los cual puede ser positivo si en los giros hay poco margen como ya se ha comentado, aunque si la trayectoria no está muy ajustada a los obstáculos podría ser interesante que el robot se separase algo más de la trayectoria dado que se reduciría el tiempo del trayecto. A pesar de ello, como ya se ha dicho, si la distancia seleccionada es demasiado baja, cuando trayectoria haga un giro es posible que el robot comience a oscilar alrededor del camino adecuado, aumentando enormemente el tiempo de llegada y pudiendo incluso llegar a separarse tanto que choque con algún obstáculo o pierda la trayectoria que debe seguir (Fig. 2.1).

Por lo tanto, el algoritmo funciona de la siguiente manera: en cada ciclo el robot, conociendo la trayectoria que debe seguir y su posición actual, busca un punto objetivo, que pertenece a la trayectoria y se encuentra a una distancia igual a la de "lookahead", calcula las velocidades que deben tener cada una de las ruedas para llegar al punto objetivo desde la posición actual, y comienza a avanzar. En el siguiente ciclo se debe actualizar la posición del robot, teniendo en cuenta lo que ha avanzado, para que, en caso de que el punto objetivo se encuentre ahora a una distancia menor de la de "lookahead", se busque un nuevo punto objetivo, de manera que el robot haga nuevamente los cálculos de las velocidades necesarias para llegar a él. En caso de que al comenzar un ciclo se observe que la distancia al punto objetivo sigue sin ser menor que la de "lookahead", el robot seguirá avanzando hacia ese punto con las velocidades calculadas anteriormente, hasta que se encuentre lo suficientemente cerca como para tener que buscar un nuevo punto objetivo. De esta manera, conociendo la trayectoria y la posición y orientación inicial del robot, y siendo *dist* la distancia entre el punto

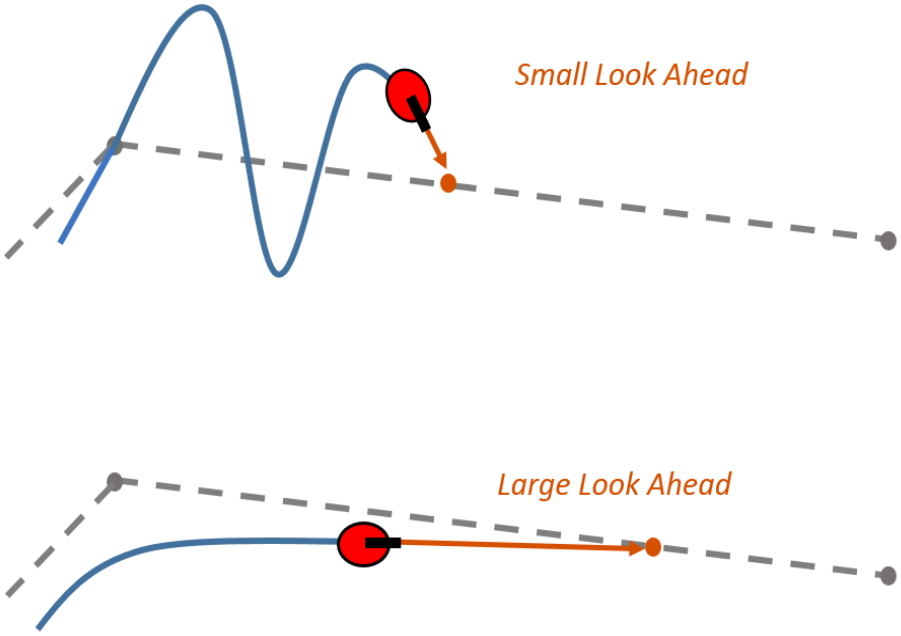

Figura 2.1: Influencia de la distancia de "lookahead"

objetivo y la posición actual del robot, el algoritmo en pseudocódigo sería:

1. Búsqueda de punto objetivo con dist mayor ó igual a distancia lookahead
2. Cáculo de velocidades en las ruedas
3. Avance a velocidad calculada
4. Actualización de posición y orientación del robot
5. Si dist mayor que distancia lookahead volver al punto 3, si no volver al punto 1
6. Repetir puntos del 1 al 5 hasta que punto objetivo = punto de destino
7. Cuando punto objetivo = punto de destino, cálculo de velocidades en las ruedas
8. Avance a esa velocidad hasta que dist menor que margen de llegada

Para calcular las velocidades a las que deben girar las ruedas para ir al punto objetivo, además de conocer la posición actual del robot como ya se ha comentado, también es necesario conocer su orientación, y es necesario conocerla en cada momento, ya que en cualquier ciclo de ejecución del programa se puede necesitar. Por ello tanto la posición como la orientación del robot deben ser actualizadas en cada ciclo, lo cual

se puede lograr mediante sensores o cámaras externos o mediante un estimador de la posición.

Uno de los métodos más comunes para calcular la curvatura que debe llevar el robot para llegar al punto objetivo consiste en trazar un círculo que contiene tanto la posición actual del robot como al punto objetivo, y cuyo centro se encuentra en el eje imaginario alrededor del cual giran ambas ruedas del robot. Una vez hecho esto, sabiendo que la curvatura de un circulo es siempre inversamente proporcional a su radio, se puede hallar fácilmente la curvatura que debe llevar el robot utilizando solamente el Teorema de Pitágoras.

Para utilizar este método de cálculo de la curvatura se debe trabajar en un sistema de referencia local con origen en la posición del robot, de manera que se debe pasar la posición del punto objetivo a este sistema. A continuación se muestra una imagen del sistema de referencia necesario.

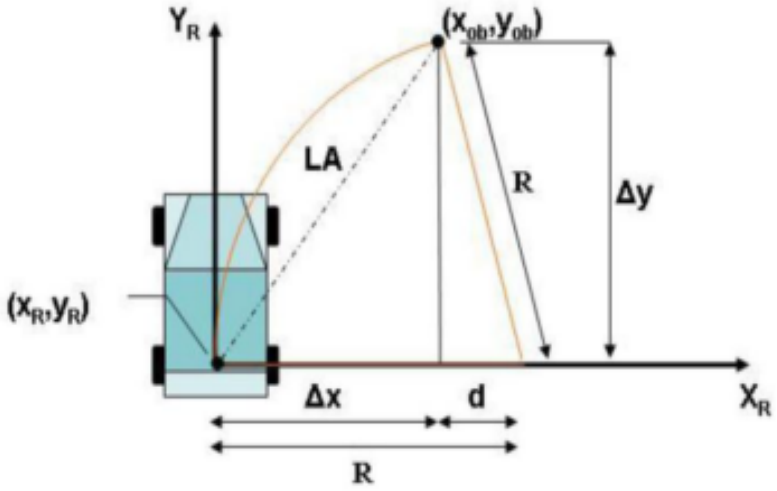

Figura 2.2: Sistema de referencia local

En la cual:

- **R:** radio de curvatura
- **LA:** distancia de "lookahead"
- **(xob,yob):** posición del ponto objetivo
- **(xr,yr):** posición del robot

A partir de esta figura, y utilizando el Teorema de Pitágoras se obtienen la expresiones:

$$(\Delta x)^2 + (\Delta y)^2 = LA^2$$

$$\Delta x + d = R$$

$$(R - \Delta x)^2 + (\Delta y)^2 = R^2$$

Al desarrollar estas expresiones y despejar la variable R se obtiene:

$$R = \frac{(\Delta x)^2 + (\Delta y)^2}{2\Delta x} = \frac{LA^2}{2\Delta x}$$

De esta forma, y teniendo en cuenta la relación entre el radio de un circulo y su curvatura que ya se ha comentado anteriormente, se obtiene:

$$\gamma = \frac{1}{R} = \frac{2\Delta x}{LA^2}$$

Como se puede observar, la curvatura que deberá tomar el robot depende solamente del desplazamiento lateral necesario para llegar al punto objetivo, un valor que irá cambiando constantemente dependiendo de la posicion del robot y del punto objetivo; y de la distancia de "lookahead", un parámetro constante a lo largo de todo el recorrido del robot.

Este es solo uno de los métodos posibles, ya que como ya se ha comentado la base del control Pure Pursuit consiste en seleccionar un punto objetivo, que el robot se dirija hacia él y cuando se haya acercado lo suficiente seleccionar un nuevo punto objetivo, todo ello sin pararse; de manera que el método de cálculo para que el robot vaya hacia el punto objetivo no tiene por qué ser siempre este, y variará en función de la aplicación y de las circunstancias en las que sea implementado el algoritmo [4].

2.2. Ventajas y desventajas del control Pure Pursuit

Actualmente el campo de los vehículos autónomos o auto guiados es de gran interés, ya que está siendo usado en muy diversas aplicaciones, desde robots pertenecientes a procesos de fabricación industrial hasta los coches sin conductor que más tarde o más temprano llegarán a ser los más comunes en el mercado, pasando por robots que realicen tareas domésticas como limpiar el suelo, o robot cirujanos que lleven a cabo operaciones sencillas o ayuden al medico.

Por este motivo, el seguimiento de trayectorias está siendo objeto de estudio y se están desarrollando nuevos métodos más precisos y seguros, para que sean utilizados en aplicaciones complejas. Es por esto por lo que un método como el pure pursuit posee gran importancia, pudiendo destacar las siguientes ventajas:

- Para empezar, puede llegar a ser utilizado en algunas aplicaciones sencillas en las que no sea requerida una gran precisión, como robots aspiradores.
- Por otro lado, puede servir como método provisional para hacer pruebas y experimentos con un sistema antes de implementar otro método mejor, ya que es sencillo y rápido de programar.
- Por último, puede ser útil para desarrollar un algoritmo mejor a partir de él, usando su estructura básica y haciendo algunas modificaciones o aportaciones extra que mejoren sus prestaciones y lo hagan adecuado para el sistema deseado [5].

A pesar de ello, este tipo de algoritmo presenta ciertas limitaciones que hacen que no sea de gran utilidad en aplicaciones complejas. Las limitaciones más importantes que presenta son:

- En primer lugar, que un controlador de este tipo no es capaz de seguir exactamente el camino entre los puntos, de manera que para obtener un comportamiento adecuado del robot será necesario ajustar los parámetros como las velocidades y la distancia de "lookahead", cosa que no es en absoluto sencilla ya que los valores óptimos variarán dependiendo del sistema y de la aplicación, y para hallarlos es necesaria mucha experimentación.

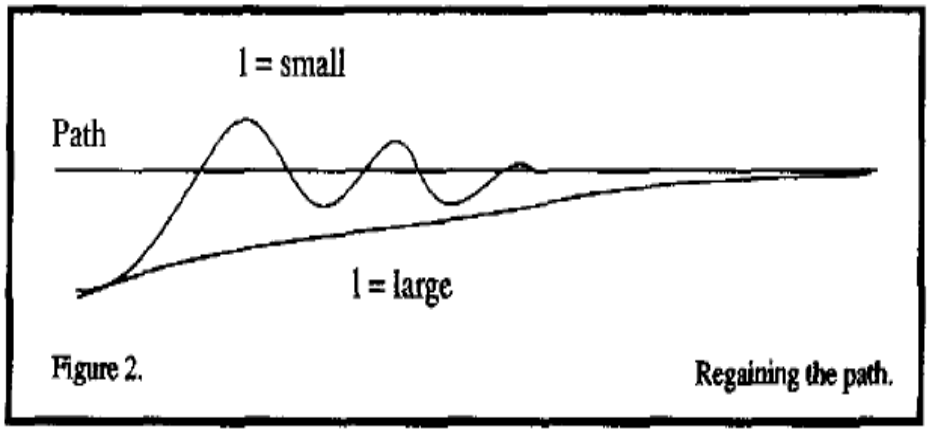

Figura 2.3: Necesidad de ajuste de parámetros

– En segundo lugar, este algoritmo no tiene en cuenta la llegada al destino, por lo que el robot no se queda estable al llegar al punto final de la trayectoria, y hace necesario implementar un margen de distancia a punto final dentro del cual pueda detenerse el robot y su posición sea aceptable como meta final [6].

2.3. Algoritmo adaptado al sistema

Como ya se ha comentado, el sistema en el que se implemente el control pure pursuit influirá en la forma de hacerlo, de manera que hay que analizar las particularidades que tiene la plataforma de robots para saber cuál es la mejor manera de desarrollar este algoritmo para nuestro robot.

En primer lugar, hay que destacar la presencia de la cámara situada sobre la base, y que reconoce tanto la posición y la orientación del robot como la de los obstáculos. Por otro lado, el destino debe ser introducido manualmente por el usuario, y teniendo tanto la posición inicial del robot como la de los obstáculos, el sistema calcula una trayectoria. A continuación, mediante el uso de señales wifi, se le transmite al robot tanto su posición y orientación inicial como cada uno de los puntos que componen la trayectoria, todo ello en un sistema de coordenadas común, que tiene su origen en una de las esquinas de la base.

De esta manera, queda claro que el robot posee todos los puntos que componen la trayectoria desde que comienza a moverse, que era uno de los requisitos para poder utilizar un algoritmo Pure Pursuit, por lo que ahora es necesario centrarse en el otro requisito, que el robot conozca su posición en todo momento. Debido a la construcción del sistema, la cámara se encuentra en todo momento sobre la base, por lo que esta

es capaz de detectar tanto la posición como la orientación del robot en cada instante, datos que el sistema es capaz de mandar el robot por señales wifi igual que al comienzo.

No obstante, esta opción no es la más adecuada, a pesar de que al localizar al robot con la cámara se obtenga su posición de manera muy exacta, dado que el envío y la recepción de los paquetes de datos mediante señales wifi es un proceso delicado y lento, y se busca que el robot siga la trayectoria de manera fluida y sin pausas. Si el robot hiciera una pausa cada vez que llega a uno de los puntos de la trayectoria, se dispondría de tiempo suficiente para que el sistema le enviase su posición y orientación actual mediante wifi, pero como en un algoritmo Pure Pursuit se debe disponer de la posición y orientación actualizadas en cada ciclo, si se llevara a cabo de esta manera se ralentizaría enormemente el avance del robot.

Es por este motivo por el que se ha decidido que en el instante inicial, el sistema le mande su posición y orientación actual al robot, así como todos y cada uno de los puntos que componen la trayectoria, y que para actualizar la posición y orientación del robot en cada ciclo se va a utilizar un estimador interno. Dicho estimador se encuentra implementado en el código interno del robot ARDUINO, y a partir de la posición y orientación inicial que recibe el robot mediante wifi va calculando cómo se mueve y avanza. Su funcionamiento es muy sencillo, una vez que se calcula la velocidad a la que debe ir cada una de las ruedas para llegar al punto objetivo, cosa que se hace en cada ciclo, el estimador sabe qué velocidad le transmite cada motor a cada rueda y durante cuánto tiempo, y con estos datos puede saber qué movimiento tendrá el robot, por lo que actualizar la posición conociendo la posición anterior resulta trivial.

Sin embargo, para ajustar este estimador es requerida una gran cantidad de experimentación, ya que a la hora de fijar las velocidades para los motores lo que se les transmite es un potencial, entre el máximo y el mínimo que admite el dispositivo, y dependiendo del peso del robot o de la cantidad de batería restante, con un mismo potencial, avanzará con una velocidad mayor o menor, y lo mismo pasa con las velocidades de giro. Así, para que el estimador tenga en cuenta una velocidad similar a la velocidad real a la que se mueve el vehículo, se ha ajustado los datos mediante prueba y error, por ejemplo, si el robot comenzara un giro antes de lo que debería es que la velocidad real a la que avanza es menor que la que tiene en cuenta el estimador, por lo que es necesario aumentar una o disminuir la otra, y los mismo para los giros tanto hacia la derecha como hacia la izquierda.

Ahora que ya disponemos de la trayectoria completa y de la posición y orientación del robot en cada instante contamos con todo lo necesario para implementar el controlador. Observando el funcionamiento del sistema, vemos que las trayectorias fijadas se van a tratar en su totalidad de lineas rectas, que unen cada uno de los puntos que el sistema ha enviado por wifi al robot y que componen la trayectoria. Teniendo esto en mente, es fácil llegar a la conclusión de que una vez que el robot se encuentre correctamente orientado desde uno de los puntos de la trayectoria hacia el punto consecutivo, no será necesario que gire hasta que llegue a dicho punto, para comenzar a buscar el siguiente. De esta manera, se puede evitar la necesidad de que el robot busque un nuevo punto objetivo constantemente, dado que una vez que se encuentre avanzando por la trayectoria en línea recta todos los puntos objetivos que sean seleccionados estarán también siguiendo la misma linea recta, hasta que se llegue a uno de los puntos donde se unen dos rectas.

Por este motivo, y a fin de evitar cálculos innecesarios que puedan ralentizar el funcionamiento del robot, se ha decidido que una vez que el robot este en la línea recta adecuada continúe avanzando sin girar hasta que se acerque a una determinada distancia del punto al que se dirigía, que será la distancia de "lookahead", momento en el cuál calculará el giro necesario para alinearse con la siguiente recta que lo llevará al siguiente punto objetivo. Así, los únicos puntos objetivos que irá teniendo el robot serán los que le ha mandado el sistema como *puntos clave* de la trayectoria o nodos, ya que el resto de la trayectoria solo esta compuesta por las rectas que van uniendo estos nodos consecutivamente, y no es necesario ir seleccionando puntos objetivos nuevos a lo largo de la recta.

En consecuencia, el funcionamiento del robot será el siguiente: en primer lugar recibirá su posición y orientación actual y los nodos de la trayectoria mediante wifi, a continuación, girará sobre sí mismo para orientarse hacia el primer nodo, que será su primer punto objetivo, y una vez hecho esto comenzará a avanzar en linea recta hasta llegar a una determinada distancia del nodo, la distancia de "lookahead" fijada anteriormente; en ese momento su nuevo punto objetivo pasará a ser el siguiente nodo de la trayectoria, y el robot calculará las velocidades necesarias en cada motor para converger a la línea recta que une el nodo anterior y el nuevo punto objetivo mediante un control PI. Dado que este cálculo es muy rápido, el robot no necesita pararse, de forma que comienza a girar con las velocidades calculadas hasta colocarse en la línea recta, y una vez que llega a ella continúa avanzando sin girar hasta que vuelve a llegar a la distancia de "lookahead" del punto objetivo, momento en el que se repite

la operación anterior. Esto continua repitiéndose hasta que el punto objetivo es el destino final, momento en el que el robot se acerca a él hasta una distancia igual a la de "lookahead" o menor, dependiendo de cual se haya fijado, a fin de que la posición final sea adecuada, y en ese momento se detiene.

Dado que la plataforma divide la base en celdas cuadradas de 25 cm de lado para calcular la trayectoria que debe seguir el robot para evitar los obstáculos, se ha decidido fijar como distancia de lookahead 6 cm, ya que con esta distancia el robot sigue estando dentro de la celda correcta, y para mejorar la precisión de llegada al destino final se ha fijado que el robot debe acercarse al último punto hasta una distancia de 3 cm. Por otro lado, para calcular las velocidades necesarias para los giros, se tienen en cuenta dos parámetros acumulativos, en primer lugar uno llamado "large" y que cuantifica la distancia al punto objetivo, y en segundo lugar "gir", que tiene en cuenta la diferencia angular entre la orientación del robot y la recta que debe seguir. Multiplicando estos parámetros por unas constantes obtenidas de la experimentación se obtienen las velocidades de cada una de las ruedas. Cabe destacar que si la velocidad calculada para el motor es menor de un cierto umbral (en este caso 50 siendo 255 el máximo del PWM) dicho motor no es capaz de hacer que la rueda gire debido a la fricción seca en el eje motor, y por este motivo, si la velocidad calculada sale por debajo de este límite, se aumenta hasta un valor que sí que permita girar a la rueda.

Antes de comenzar este proyecto, este sistema contaba con dos controladores ya implementados para dirigir el movimiento del robot. Por un lado, el control más sencillo, el cual una vez recibida la trayectoria hace que el robot gire sobre sí mismo en el punto de partida hasta orientarse hacia el primer punto al que debe dirigirse. En ese momento, con el robot orientado, éste comienza a avanzar en línea recta hasta llegar al punto de la trayectoria al que se dirigía, y una vez ha llegado vuelve a detenerse y girar sobre sí mismo hasta volver a quedar orientado hacia el siguiente punto. Esta operación se repite, parándose en cada punto, orientándose hacia el siguiente y avanzando en línea recta hasta él, hasta que el robot se encuentra sobre el punto de destino. Por otro lado, se puede encontrar el control Proporcional Integral (PI). Utilizando este controlador, el robot se detiene al llegar a cada uno de los puntos de la trayectoria, igual que con el anterior, pero existe una pequeña diferencia, y es que, si el error entre la orientación que tiene el robot al detenerse y la que debe llevar para dirigirse al siguiente punto es mayor de 15 grados, gira sobre sí mismo hasta orientarse igual que en el caso anterior, pero si este error es menor de 15 grados, el robot comienza a avanzar con esta orientación y va girando mientras avanza hasta

volver a situarse sobre la recta que llega al siguiente punto de la trayectoria, y a partir de ahí sigue la recta.

La principal ventaja que presenta este control Pure Pursuit frente a los otros dos que estaban implementados para este robot y esta plataforma es que el robot no tiene la necesidad de detenerse en ningún momento, de manera que recorre la trayectoria de manera mucho más rápida, ya que no hace los giros sobre sí mismo ni quedándose en en sitio, sino que gira mientras sigue avanzando. Queda claro pues que este controlador tiene como ventaja una mayor rapidez dado que el robot no se para en ningún momento desde que comienza su recorrido hasta que llega al destino, pero su inconveniente es que no sigue la trayectoria de manera tan exacta como los anteriores, sino que en los giros no llega al vértice. Esto no es un problema en la plataforma que trabajamos, sabiendo que las celdas son de 25 por 25 centímetros y por lo tanto los giros no están muy ajustados a los obstáculos, además de que reduciendo la distancia de "lookahead" se seguiría la trayectoria de forma mucho más precisa, pero en aplicaciones o entornos en los que fuera requerida una precisión mucho mayor quizá un control de este tipo no fuera el más óptimo, o en todo caso sería necesario ajustar mucho más los parámetros para seguir la trayectoria de manera muy estrecha. Aun así, para evitar que el robot se desvíe demasiado y como consecuencia el punto de destino no sea el esperado o incluso pueda colisionar con un obstáculo en trayectorias complejas, se ha incluido una orden que hace que, en el caso de que el robot tenga un error mayor de 5 grados con respecto a la recta que esta siguiendo en ese momento, se reoriente hacia el punto objetivo sin pararse, de manera que evitamos que un pequeño error de orientación o una ligera diferencia en el comportamiento de los motores en recta hagan que el robot se vaya desviando cada vez más de la trayectoria inicial.

Capítulo 3

Simulaciones en el software Matlab

3.1. Funcionamiento de la toolbox

Con el fin de entender de qué manera cambia el comportamiento de un control Pure Pursuit con la variación de sus parámetros más representativos, se van a realizar una serie de simulaciones con diferentes valores en los mismos y comentar las diferencias que se observen en los resultados. Asimismo, se va a comparar el comportamiento que tendría el robot en una misma situación con un control pure pursuit y con un control PI para poder ver qué ventajas y desventajas presenta cada uno y en que situaciones puede ser más optima su utilización.

Se ha decidido hacer simulaciones con un software informático, ya que si se quisieran hacer en la plataforma de robots real sería mucho más costoso cambiar los parámetros y sobretodo sería mucho más difícil medir los resultados, ya que utilizando la plataforma lo único que se obtiene es el movimiento del robot, mientras que en un software se pueden obtener diferentes datos y gráficas de nuestro interés. Si se deseara hacer la experimentación con la plataforma real no se podría medir de manera precisa la trayectoria seguida ni la fijada inicialmente, así como los datos de las velocidades de cada rueda en cada momento o la orientación.

El software elegido ha sido Matlab, ya que se trata de un programa informático fácil de usar y muy versátil, pero que a la vez proporciona una gran capacidad de simulación y posibilidad de representar gráficamente cualquier conjunto de datos. Además, este software dispone de la capacidad de utilizar "toolbox" ya desarrolladas por otros usuarios, de manera que facilita enormemente cualquier trabajo y nos evita la necesidad de tener que partir de cero para realizar el calculo o la simulación deseada. En este caso se va a utilizar una toolbox diseñada por el Área de Ingeniería de Sistemas y Automática de la Universidad de Zaragoza, llamada "Robot Motion

Toolbox" (http://webdiis.unizar.es/RMTool/) y diseñada para estudiar el movimiento de robots móviles en un plano de dos dimensiones.

Una vez descargados los archivos e incluidos en Matlab, se ha de ejecutar la orden "rmt" y se abrirá una interfaz. En ella encontraremos un plano o mapa a la derecha, debajo de él un espacio para diferentes gráficas que aparecerán una vez obtengamos los resultados de la simulación, y en la zona izquierda la posibilidad de cambiar los valores de diferentes parámetros y opciones. En la figura 3.1 se pueden ver las diferentes opciones que se pueden modificar para llevar a cabo la simulación. En primer lugar es necesario seleccionar el tipo de simulación que se va a llevar a cabo: en nuestro caso será la consistente en ir de un punto de inicio a un punto de destino sin pasar por unos determinados obstáculos, aunque hay otras opciones existentes como la de dividir el mapa en diferentes regiones por las que debe o no debe pasar el robot.

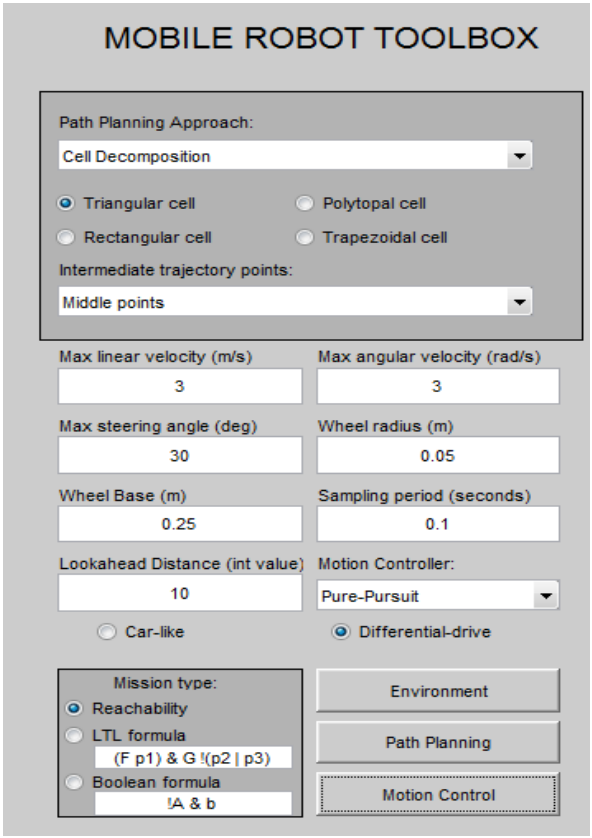

Figura 3.1: Diferentes opciones disponibles para la simulación

A continuación, se debe pulsar el botón "Environment" para definir los obstáculos. En primer lugar se selecciona el número de obstáculos deseado, y una vez hecho esto se deben definir los obstáculos en el mapa. Cada vez que se haga click con el botón izquierdo del ratón se colocará un vértice del polígono que conformará el

obstáculo, y al hacer click con el botón derecho se indica que se trata del último vértice de ese obstáculo. Esto habrá que repetirlo tantas veces como obstáculos se hayan seleccionado, y después se deberá colocar el punto de partida y el punto de destino, ambos con el botón derecho del ratón. Posteriormente se debe seleccionar el método que se va a utilizar para calcular la trayectoria desde el punto inicial hasta el destino sin pasar por los obstáculos; de los diferentes métodos posibles se va a utilizar el de descomposición triangular, ya que proporciona una trayectoria formada por líneas rectas, al igual que la plataforma en la que se coloca el robot real, y además se trata del método más sencillo, y teniendo en cuenta que el objetivo del trabajo no es el cálculo de trayectorias sino su seguimiento resulta un método adecuado. Seleccionado el método, se pulsa el botón "Path planning" y aparecerá la trayectoria que ha calculado el programa y deberá seguir el robot dibujada en el mapa.

Por último, solamente queda fijar algunos últimos parámetros y opciones referentes al robot y ya se podrá pulsar en el botón de realizar simulación. Lo primero será fijar el tipo de robot con el que se desea llevar a cabo la simulación, ya que disponemos de dos modelos diferentes: "car-like" y "differential-drive". Como su propio nombre indica, el modelo "car-like" es similar a un automóvil, siendo sus principales características que ambas ruedas girarán a la misma velocidad y que tendrá un límite de grados de giro posibles, ya que simula un eje que une ambas ruedas y posee un diferencial. Este límite de grados de giro es uno de los parámetros que es posible fijar en la simulación, siendo 30º el máximo valor posible. El modelo "differential-drive" es el elegido para las simulaciones, ya que se trata del que más se asemeja al robot real para el que hemos implementado el control, y sus principales características radican en que cada rueda es capaz de girar a una velocidad diferente, al igual que el robot que posee dos motores, y que por lo tanto tiene la posibilidad de girar sobre sí mismo sin cambiar de posición, de manera que no es necesario fijar un valor límite de ángulo de giro.

Una vez seleccionado el modelo de robot, se proporcionan los valores numéricos para los diversos parámetros disponibles. Estos son: máxima velocidad de avance lineal, máxima velocidad angular o de giro, máximo ángulo de giro posible, que como ya se ha comentado anteriormente solo afecta en el caso de que se haya seleccionado el modelo de robot "car-like"; el radio de las ruedas del robot, la distancia entre ejes, el periodo de muestreo y por último la distancia de "lookahead", que solo afecta si se utiliza el control pure pursuit. Se han fijado unos valores para estos parámetros similares a los que posee el robot real para el que se ha implementado el controlador, de forma que al realizar simulaciones en situaciones similares a las hechas en la

plataforma real se obtengan resultados parecidos. Por este motivo, los valores fijados para los parámetros son 3 m/s de velocidad lineal máxima, 5 rad/s en la máxima velocidad angular, 30º para el máximo ángulo de giro posible, aunque como ya hemos dicho al no realizar ninguna simulación con el modelo de robot "car-like" este parámetro no resulta relevante; 0'05 m de radio para las ruedas, 0,25 m de distancia entre ejes, un periodo de muestreo de 0,1 segundos y una distancia de "lookahead" de 20. El valor de la distancia de "lookahead" no tiene unidades ya que se refiere al número de puntos hacia delante de su posición actual que mira el robot, pero su funcionamiento es similar a si se tratase de una distancia. Estos valores de los parámetros no se ajustan exactamente a los del robot real utilizado, pero dado que los valores que se admiten para la simulación en cada parámetro están limitados por un valor máximo y un mínimo, se han intentado fijar valores lo más similares posibles a la realidad y que guarden la misma proporción entre ellos que los del robot real. Así pues, la siguiente tabla reune los valores fijados para los distintos parámetros (3.1).

Parámetro	Valor
Max vel lineal	3 m/s
Max vel angular	5 rad/s
Max ángulo de giro	30º
Radio de ruedas	0'05 m
Dist entre ejes	0'25 m
Periodo muestreo	0'1 s
Dist de lookahead	20

Tabla 3.1: Valores fijados para cada parámetro

Como última opción, se puede seleccionar el tipo de controlador que va a utilizar el robot en la simulación, teniendo para elegir entre un control PI y un control Pure Pursuit. Se han revisado los algoritmos que rigen el movimiento del robot en la simulación para cada uno de los dos tipos de control, a fin de comprobar que su funcionamiento es correcto y similar al de los controladores reales para el robot en la plataforma. Teniendo esto en cuenta, se van a realizar una serie de simulaciones replicando las situaciones reales en las que se ha experimentado con el robot, utilizando para un mismo caso, es decir, para una misma posición de los obstáculos y de los punto de partida y de destino, el control PI y el control Pure Pursuit, con el objetivo de observar cómo varía el comportamiento del robot con cada uno de los controladores y en qué situaciones es mejor utilizar uno u otro, o cuál presenta un mejor comportamiento en general. Asimismo, se elegirá un caso representativo y en él se utilizará el control Pure Pursuit pero variando algunos de los parámetros en cada simulación para poder

observar cómo afecta cada uno de ellos al comportamiento del robot, y cuáles son los valores ideales o los más recomendados para cada situación.

3.2. Control PI versus control Pure Pursuit

En primer lugar se van a llevar a cabo una serie de simulaciones intentando replicar los experimentos hechos en la plataforma de robots real. Para ello, se han fijado los valores de los parámetros anteriormente citados, intentando que las características del robot en la simulación sean lo más parecidas posible a las del robot real, teniendo en cuenta las limitaciones de la herramienta de simulación. Además, se han colocado tanto el punto de partida como el de destino en las mismas zonas del plano que se habían fijado anteriormente en los experimentos con la plataforma real, y se han diseñado los obstáculos con la forma y en el lugar apropiado para tapar las mismas zonas del plano, de manera que en el momento que pulsemos el botón de calcular la trayectoria en la simulación, obtengamos una similar a la que ha calculado anteriormente la plataforma para que fuera mandada por señales wifi al robot real. Una vez se tiene esto, es posible llevar a cabo la simulación, con esta misma disposición de los obstáculos y puntos de partida y destino, una y otra vez pero cambiando valores de parámetros y el tipo de control utilizado en el robot, de manera que se puede trabajar en un mismo entorno y ver cómo varía el comportamiento al cambiar una determinada característica del robot.

Gracias a esto, para cada uno de los casos se va a llevar a cabo la simulación utilizando los dos tipos de control posibles: tanto el Pure Pursuit como el PI. En primer lugar, se va a empezar utilizando el control Pure Pursuit, con todos los valores de los parámetros tal y como se han fijado anteriormente. Una vez hecha la simulación, se cambiará al controlador PI, sin cambiar ningún parámetro, para comparar los resultados.

En los resultados obtenidos se pueden encontrar, en el plano donde se han colocado los obstáculos, dos líneas que van desde el punto de partida al punto de destino. La línea de color rojo es la trayectoria calculada por el algoritmo de planificación de trayectorias elegido, y que debe seguir el robot en la simulación, y la línea de color negro se trata del recorrido efectuado por el robot simulado. También se puede ver que se han completado las gráficas que representan la orientación, la velocidad y el ángulo de giro frente al tiempo. Como se puede observar en las figuras 3.2 y 3.3, el comportamiento del robot utilizando el control Pure Pursuit es apropiado, ya que sigue de manera bastante fiel la trayectoria, mientras que usando el control PI salta

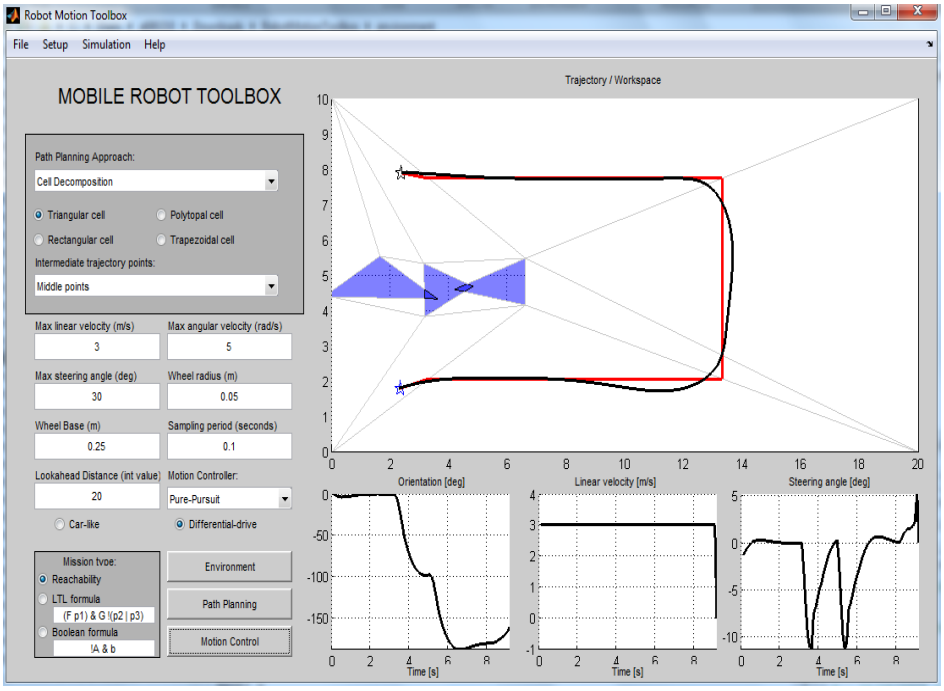

Figura 3.2: Situación 1, control Pure Pursuit V=3

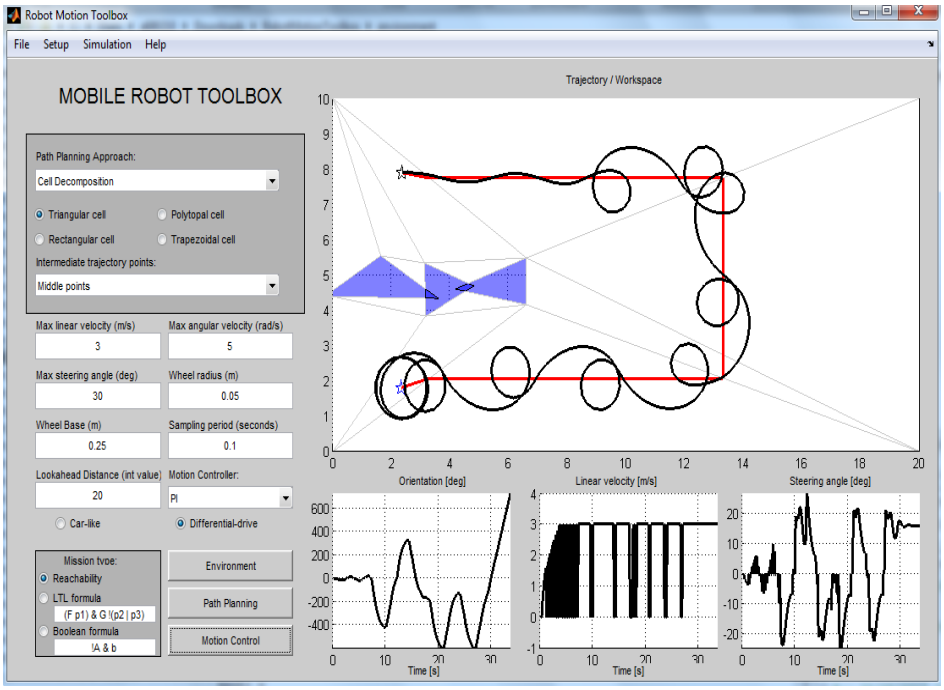

Figura 3.3: Situación 1, control PI v=3

a la vista que sobreoscila, separándose de la trayectoria fijada y perdiendo mucho tiempo, y además no consigue llegar al punto de destino de manera precisa. Para intentar mejorar el comportamiento del robot al utilizar este tipo de control, se ha decidido cambiar el parámetro de velocidad lineal máxima, fijando un valor menor al seleccionado anteriormente para intentar disminuir la oscilación.

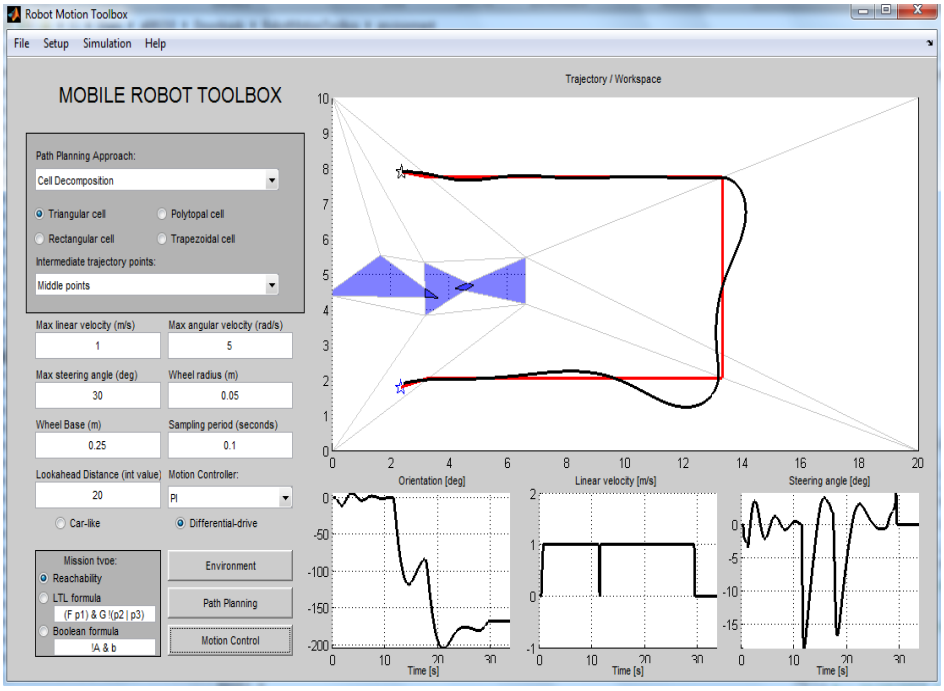

Figura 3.4: Situación 1, control PI v=1

Con el cambio realizado, el robot sigue la trayectoria de manera más estrecha y llega al punto de destino de manera más precisa, por lo tanto presenta un comportamiento mucho más adecuado. Sin embargo, en las zonas donde el robot debe girar, continúa apareciendo una oscilación mayor de la que se observa al usar un control Pure Pursuit, y además al haber disminuido la velocidad máxima lineal para mejorar el seguimiento de la trayectoria, el tiempo que tarda el robot en llegar al destino es superior a 30 segundos como se puede ver en culaquiera de las tres gráficas disponibles, mucho mayor que al utilizar el control Pure pursuit que tardaba alrededor de 9 segundos. Se han llevado a cabo simulaciones similares a las de este caso en dos casos diferentes más, cambiando los mismos parámetros y obteniendo resultados parecidos. (Ver ANEXO A)

3.3. Análisis de parámetros

A continuación, se ha seleccionado un caso a partir del cual se va a analizar la influencia de los parámetros en el comportamiento del robot. Para ello, se va a llevar a cabo la simulación para dicho caso utilizando el controlador del tipo Pure Pursuit y fijando los valores de los parámetros a los más aproximados al robot real, que ya se han comentado anteriormente. Una vez obtenidos los resultados de esta simulación, sin

variar la posición de los obstáculos ni los puntos de partida y de destino, es decir, sin variar la trayectoria que debe seguir el robot, se van a cambiar algunos de los valores de determinados parámetros clave a fin de comparar los resultados con los ya obtenidos usando los parámetros anteriores. Los parámetros que se ha decidido estudiar, dado que se consideran los más influyentes en el comportamiento del robot, son: la velocidad lineal, la velocidad de giro y la distancia de "lookahead". Lógicamente, parámetros como el radio de las ruedas o la distancia entre ejes también influyen de gran manera sobre el comportamiento que tendrá un robot, pero éstos vienen determinados por el robot con el que se vaya a trabajar, no tienen ninguna relación con el método de control de movimiento implementado en el robot, de manera que no son parámetros de interés para la experimentación en relación al objetivo de este trabajo.

En los experimentos llevados a cabo anteriormente, con el objetivo de comparar el control PI con el Pure Pursuit, la diferencia en los resultados era evidente a simple vista, ya que únicamente con la representación sobre el plano de la trayectoria seguida se podía observar claramente las diferencias que presentaba el comportamiento del robot, y si era necesario se podía consultar los datos de las gráficas generadas, como el tiempo transcurrido o las velocidades del robot en cada instante. Esto era así ya que al cambiar el tipo de control del robot, lo que se estaba haciendo era cambiar el método de seguimiento de la trayectoria, con lo cual el recorrido efectuado por el robot en cada caso difería enormemente de los otros. Esto será diferente para los experimentos que se van a llevar a cabo a continuación, ya que en ellos no se va a cambiar el método de seguimiento de la trayectoria en ningún momento, en todos el robot de la simulación utilizará un control Pure Pursuit, de manera que los cambios que se puedan observar en el recorrido efectuado por el robot en cada experimento se deberán enteramente al cambio del valor de un determinado parámetro. Dado que a priori no se conoce cómo y cuanto influirá cada uno de los parámetros, aunque se puede prever que el más significativo será la distancia de "lookahead", se ha decidido añadir una serie de comandos al algoritmo de control Pure Pursuit que utiliza la toolbox para las simulaciones, de manera que al obtener los resultados de una se nos muestre también en la ventana de comandos de matlab un nuevo dato: el error cuadrático medio entre la trayectoria que debe seguir el robot inicialmente y el recorrido efectuado por éste. Así, si hay ocasiones en las que a simple vista el recorrido representado en el plano es similar, disponemos de un dato más para poder observar de qué manera el seguimiento de la trayectoria es más preciso.

3.3.1. Cálculo del error cuadrático medio

A continuación, se va a explicar cómo se ha calculado este error cuadrático medio. En el algoritmo del control Pure Pursuit implementado en la toolbox se dispone de un vector en el que aparecen todos los valores de x, es decir el eje horizontal, de los puntos que componen la trayectoria inicial, al igual que otro vector con los valores de y, el eje vertical. Esto mismo es también fácil de conseguir para el recorrido efectuado por el robot. Por otro lado, existe una orden en matlab que compara dos vectores del mismo tamaño y calcula automáticamente el error cuadrático medio entre ellos. Por lo tanto, utilizando dicha orden se calculará el error cuadrático medio en x por un lado y en y por otro lado de la siguiente forma:

$$ECMx = \frac{1}{n}\Sigma(xrecorrido - xtrayectoria)^2$$

$$ECMy = \frac{1}{n}\Sigma(yrecorrido - ytrayectoria)^2$$

Y para obtener el error cuadrático medio total existente entre la trayectoria fijada y el recorrido efectuado por el robot en la simulación solamente será necesario utilizar el Teorema de Pitágoras para sumar los errores tanto en x como en y:

$$ECMtotal = \sqrt{ECMx^2 + ECMy^2}$$

De esta manera, el problema que nos encontramos es que el vector de la trayectoria de referencia y el del recorrido real no tienen el mismo tamaño; como es lógico, dado que el recorrido real intenta seguir la trayectoria pero en determinados momentos se separa de ella para posteriormente volver a unirse, de manera que es más largo; por lo que es necesario igualar su tamaño para poder compararlos. Para este objetivo, se calcula la cantidad de puntos que hay que eliminar del vector más largo haciendo la proporción entre ellos. Así, cada n puntos del vector más largo eliminaremos uno, de manera que no se perderá una gran cantidad de información de la trayectoria. Dado que al calcular la proporción ésta ha sido redondeada, es posible que al final de esta operación exista una diferencia de uno o dos elementos en el tamaño de los vectores, lo cual será solucionado añadiendo al vector más corto el número de elementos necesarios repitiendo el último valor que aparecía en dicho vector. Una vez hecho esto, utilizando la orden de matlab es inmediato obtener la diferencia cuadrática

media entre la trayectoria y el recorrido tanto para x como para y, de manera que lo único que resta por hacer es utilizar el Teorema de Pitágoras con estos dos valores para obtener el error cuadrático medio entre la trayectoria de referencia y el recorrido real. Cabe destacar que este valor no tiene ninguna medida, se trata solamente de una comparación entre puntos. De esta forma, queda claro que la situación en la que aparezca un mayor error cuadrático medio será aquella en la que menos preciso sea el seguimiento de la trayectoria. El algoritmo usado para calcular el error cuadrático medio aparece en la figura 3.5 .

```
[w1,l1] = size(y);
 [w2,l2] = size(yref);
 diffy = l1 - l2;
 gg = round(l1/diffy);
 removalList = 1:gg:l1;
 h = y;
h(removalList) = [];

[w3,l3] = size(h);

while(l3 ~= l2)
    if(l3 > l2)
        temp = yref(end);
        yref = [yref temp];
        [w2,l2] = size(yref);
    else
        temp = h(end);
        h = [h temp];
        [w3,l3] = size(h);
    end
end

    size(h)
    size(yref)
    RMSE1 = sqrt(mean((h - yref).^2))
```

```
    [w1,l1] = size(x);
 [w2,l2] = size(xref);
 diffx = l1 - l2;
 gg = round(l1/diffx);
 removalList = 1:gg:l1;
 h = x;
h(removalList) = [];

[w3,l3] = size(h);

while(l3 ~= l2)
    if(l3 > l2)
        temp = xref(end);
        xref = [xref temp];
        [w2,l2] = size(xref);
    else
        temp = h(end);
        h = [h temp];
        [w3,l3] = size(h);
    end
end

    size(h)
    size(xref)
    RMSE2 = sqrt(mean((h - xref).^2))
resSS = sqrt(RMSE1^2 + RMSE2^2)
```

Figura 3.5: Algoritmo utilizado para el cálculo del error

3.3.2. Experimentación

El caso base del que se va a partir utiliza el controlador Pure Pursuit y unos determinados valores para los parámetros de estudio. Una vez llevada a cabo la simulación de dicho caso, se va a variar el valor de cada uno de los parámetros de estudio, uno por uno, tanto aumentándolo como disminuyéndolo, y se va a volver a realizar la simulación para observar las diferencias existentes en los resultados. En esta ocasión, los valores que van a tener los parámetros de estudio en el caso base van a ser diferentes a los utilizados anteriormente, ya que antes se buscaba simular un robot lo más parecido posible al utilizado realmente, pero en este caso lo que se quiere es comparar los resultados al variar el valor de un determinado parámetro, tanto por encima como por debajo del valor base, y dado que algunos de los parámetros

ya estaban fijados en el valor límite que se permite en la simulación es necesario modificarlos para poder tener suficiente margen por ambos lados. Así, los valores que se van a fijar para los parámetros de estudio serán: 3 m/s para la velocidad lineal máxima, igual que en los experimentos anteriores, 3 rad/s para la velocidad angular máxima y una distancia de "lookahead" de 10. El resto de parámetros siguen manteniendo el valor fijado anteriormente. Así pues, el resultado de la simulación base se muestra en la figura 3.6 .

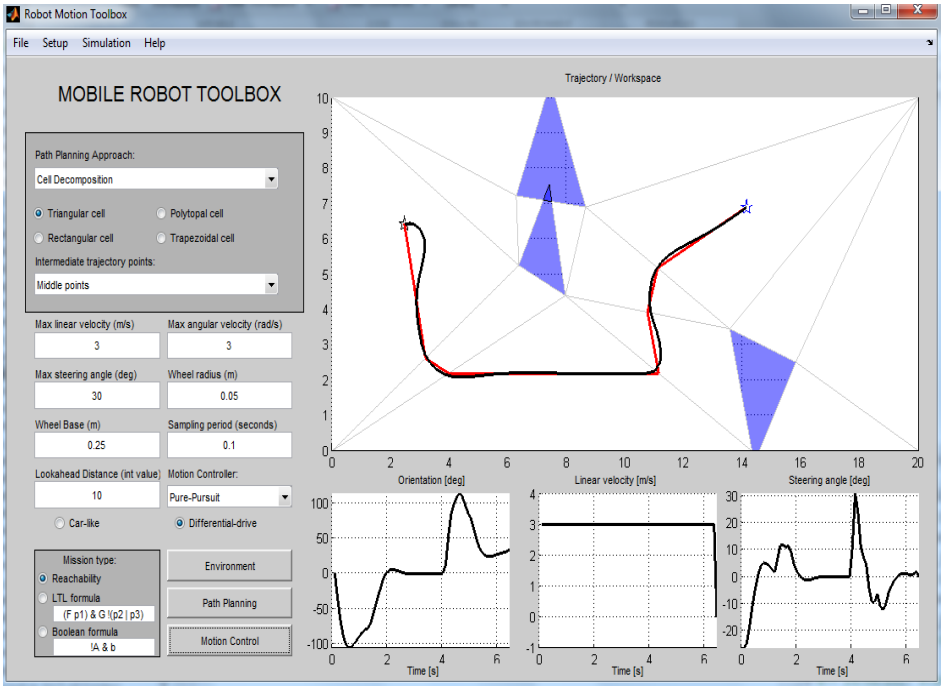

Figura 3.6: Simulación base: v=3m/s, w=3rad/s, d=10, ERROR=5'8

Se puede decir que en este caso el robot presenta un comportamiento adecuado, ya que sigue la trayectoria de manera aceptable y llega al punto de destino con bastante precisión, y presenta un error cuadrático medio de 5'8.

Velocidad lineal

En primer lugar, se va a estudiar la influencia de la velocidad lineal máxima, por lo que se van a llevar a cabo dos simulaciones, en una de ellas se aumentará el valor de este parámetro a 4 m/s y en la otra se disminuirá a 2 m/s. (Ver Figuras 3.7 y 3.8)

Aunque a simple vista el seguimiento de las trayectorias no varía en gran medida, sí que se observan dos grandes diferencias, por un lado, el error, que aumenta al aumentar la velocidad máxima y disminuye en caso contrario, lo cual coincide con

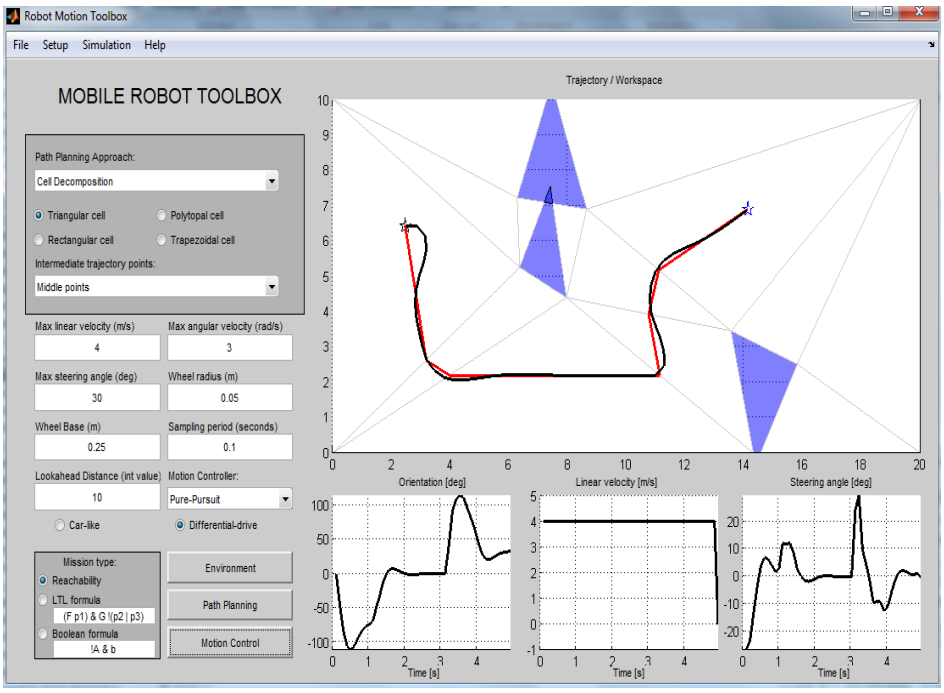

Figura 3.7: v=4m/s, w=3rad/s, d=10, ERROR=6

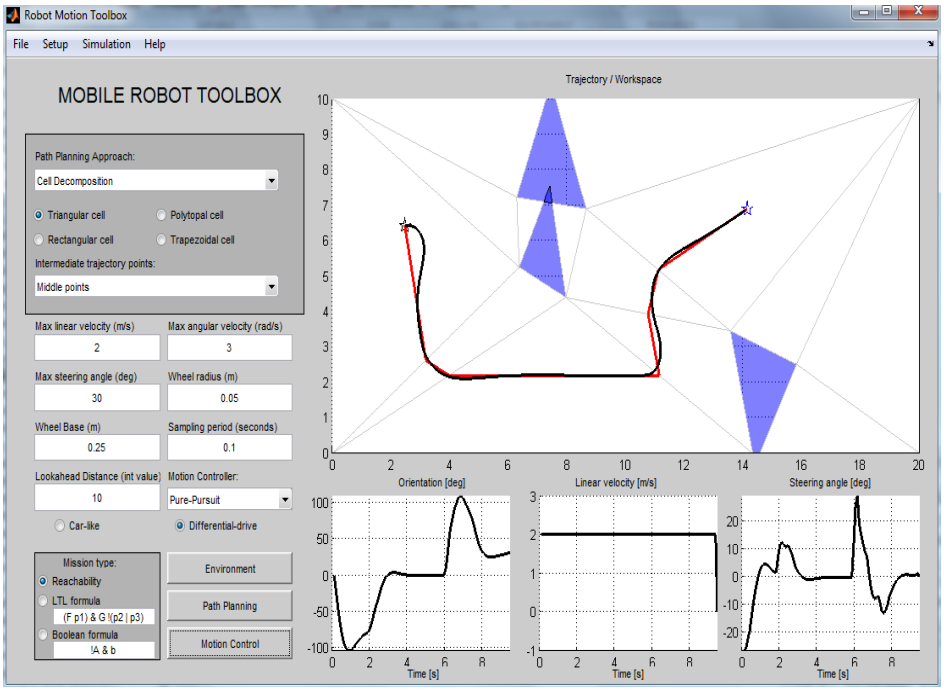

Figura 3.8: v=2m/s, w=3rad/s, d=10, ERROR=5'6

lo que se podía pensar al observar la trayectoria, de manera que a menor velocidad avance el robot menos se separará su recorrido de la trayectoria. Por otro lado, la segunda gran diferencia se encuentra en el tiempo de recorrido, que al fijar la velocidad elevada no llega a 5 segundos y en cambio al fijar un valor menor aumenta hasta más de 8 segundos, cuando seleccionando un valor de 3 m/s el tiempo era de alrededor de

6 segundos.

Velocidad angular

El siguiente parámetro que se va a estudiar será la velocidad angular máxima o velocidad máxima de giro. En esta ocasión, se va a fijar un valor de 1 rad/s. (Ver figura 3.9)

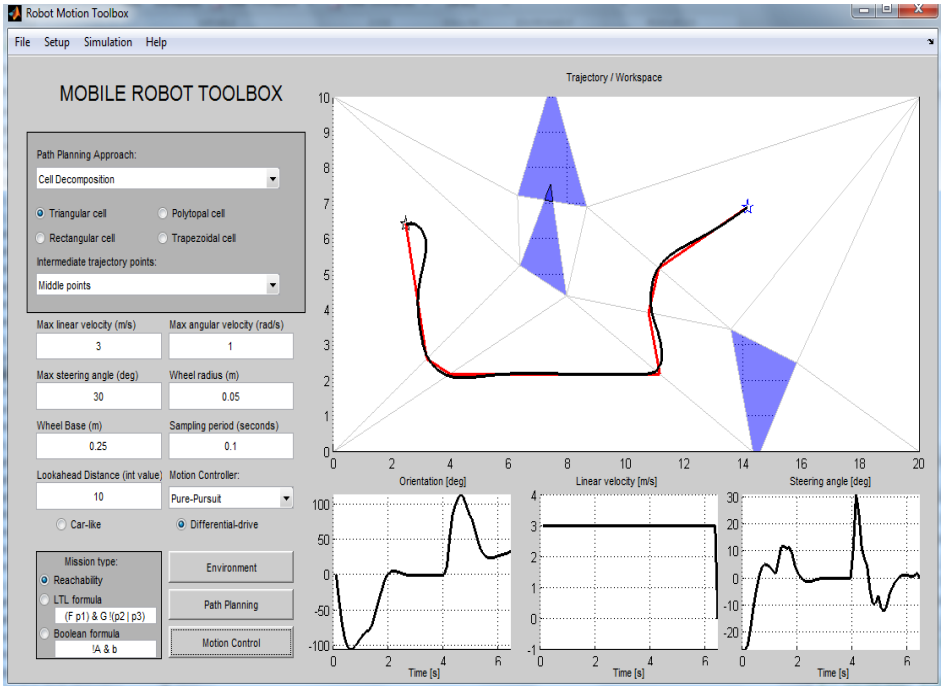

Figura 3.9: v=3m/s, w=1rad/s, d=10, ERROR=5'8

En esta ocasión, el resultado no cambia en absoluto. Esto hace pensar que el valor fijado de 3 rad/s era el máximo posible pero este máximo no llegaba a alcanzarse en ningún momento, de manera que si se aumentara este límite tampoco se encontraría cambio alguno. Por este motivo, se decide fijar valores menores a 1 rad/s para ver en qué momento comienza a afectar al resultado. (Ver Figuras 3.10 y 3.11)

Al fijar un valor de 0'5 rad/s, ya se puede observar un cambio en la trayectoria, aunque éste apenas es perceptible a simple vista, ya que la única diferencia es que al separarse de la trayectoria en una curva tarda ligeramente más en volver a ella. Por este motivo, se ve que el error ha aumentado ligeramente, ya que el seguimiento de la trayectoria empeora aunque no sea de forma muy llamativa. Con el valor de 0'1 rad/s sí que se ve claramente que el recorrido del robot deja de seguir la trayectoria casi por completo, cosa que concuerda con el gran aumento del error, y aunque al final llegue

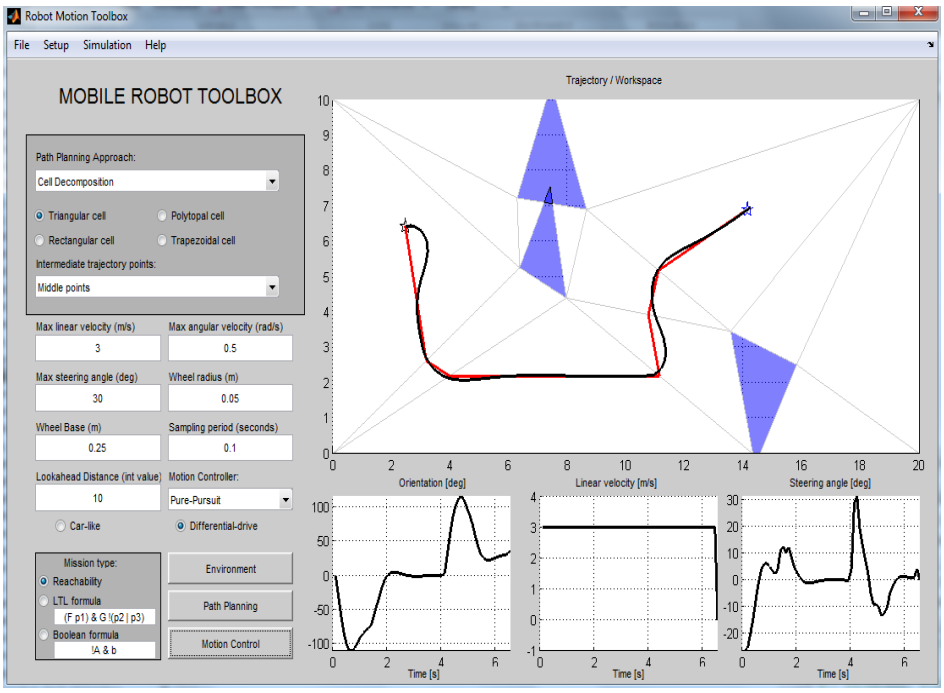

Figura 3.10: v=3m/s, w=0'5rad/s, d=10, ERROR=5'9

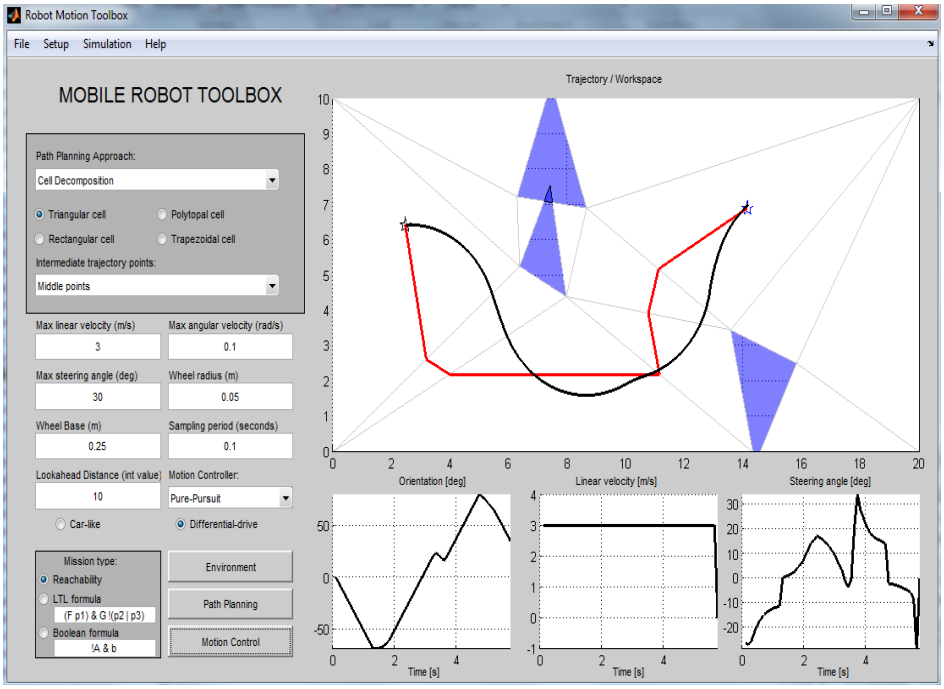

Figura 3.11: v=3m/s, w=0'1rad/s, d=10, ERROR=6'4

al punto de destino igualmente, este no sería un comportamiento adecuado del robot.

Distancia de lookahead

Por último, se va a variar el valor de la distancia de "lookahead", en primer lugar se fijará un valor de 20, y a continuación se seleccionará un valor de 5. Con estos valores, los resultados obtenidos son los siguientes: (Ver Figuras 3.12 y 3.13)

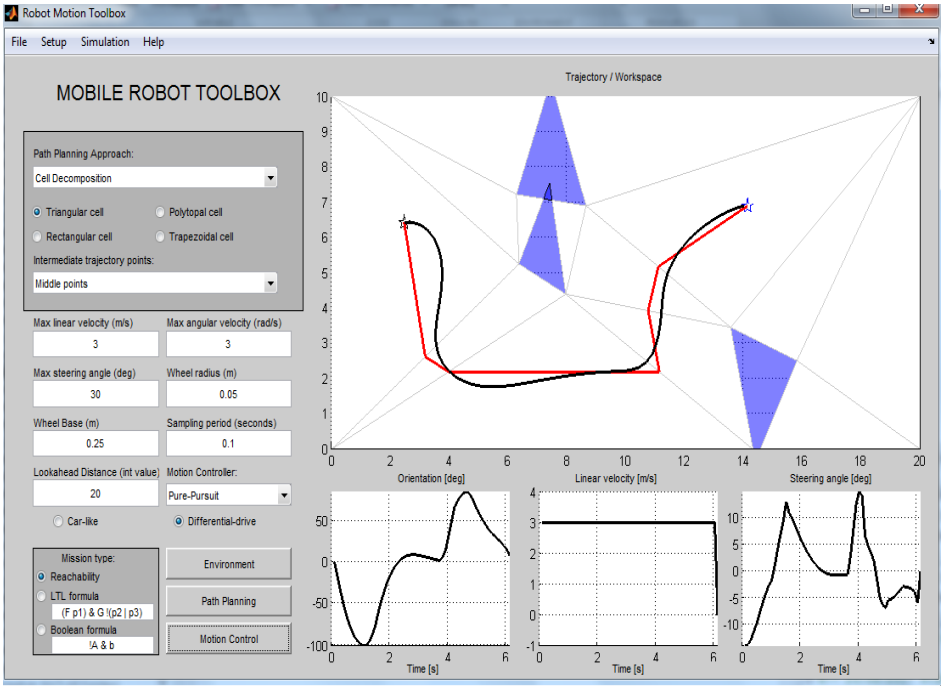

Figura 3.12: v=3m/s, w=3rad/s, d=20, ERROR=6'1

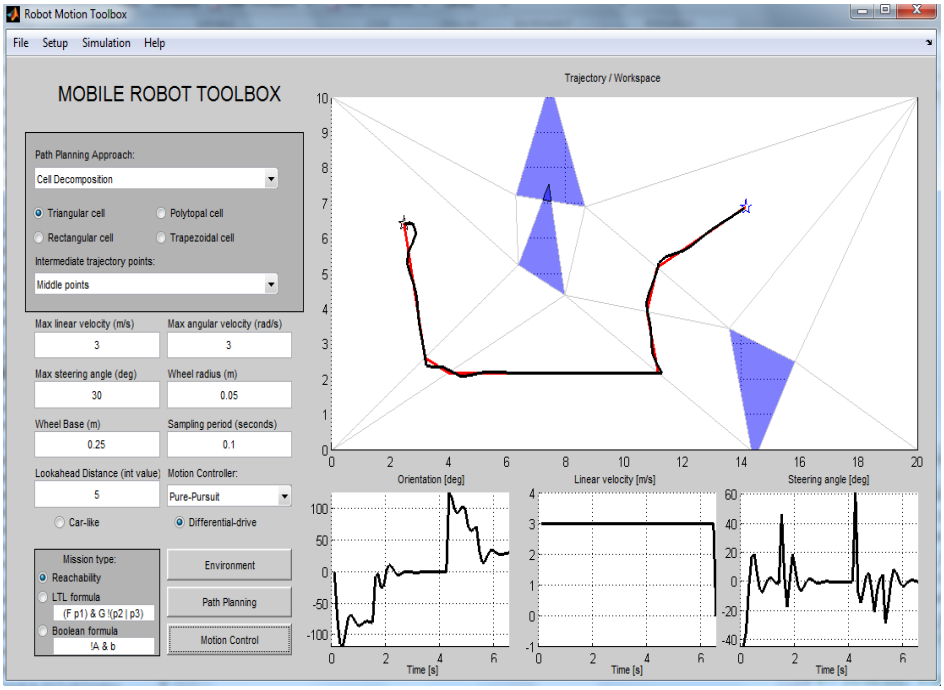

Figura 3.13: v=3m/s, w=3rad/s, d=5, ERROR=5'4

Salta a la vista que con el valor de 20 en la distancia de "lookahead" el recorrido se separa mucho más de la trayectoria de referencia, lo cual también se ve reflejado en el aumento del error, aunque el comportamiento podría seguir siendo válido para aplicaciones en las que se requiera una precisión muy baja, ya que si lo que más se valora es la velocidad, se observa que el tiempo de recorrido es ligeramente menor al del caso base. Por otro lado, al fijar el valor de 5 se ve que el seguimiento de la trayectoria de referencia es mucho más estrecho, y el error disminuye respecto al caso base con una distancia de "lookahead" mayor. Al observar que el tiempo de recorrido es prácticamente el mismo y que se disminuye el error se podría pensar que esta es la opción más adecuada, pero no tiene porqué serlo, ya que se produce una pequeña oscilación, que se observa tanto en el recorrido representado en el plano como en las gráficas de la orientación y el angulo de giro frente al tiempo. De esta forma, en este caso concreto puede que sea aceptable esta oscilación, pero en otros muchos casos puede generar grandes problemas, de manera que una distancia de "lookahead" muy baja no suele ser lo más adecuado.

Capítulo 4

Conclusiones

Una vez llevadas a cabo todas las simulaciones, del primer grupo de ellas, en las que se comparaba el control Pure pursuit con el control PI, se puede concluir que, aunque para ciertas aplicaciones podría ser válido el control PI, dada su gran sencillez, en general el control Pure Pursuit presenta características mucho más deseables, ya que el seguimiento de la trayectoria es mucho más estrecho, y presenta menor oscilación que al utilizar el control PI incluso avanzando a velocidades mucho mayores que este, de manera que permite que el tiempo de llegada sea más bajo y el recorrido más óptimo, además de que a pesar de ser algo más complejo que el controlador PI se sigue tratando de un tipo de control simple y fácil de implementar. Por otro lado, se puede observar que el control Pure Pursuit es muy versátil, ya que presenta muy buenos comportamientos en diversos casos, de lo que se puede deducir que es capaz de seguir la gran mayoría de las trayectorias sin dejar de lado los buenos resultados.

Con respecto a las simulaciones en las que se experimentaba con los diversos parámetros relevantes del controlador Pure Pursuit, se puede concluir que la velocidad de avance lineal influye mayoritariamente en el tiempo de recorrido, como es lógico, y aunque haga variar ligeramente la precisión del seguimiento de la trayectoria, siendo esta mayor a menor velocidad de avance, su influencia no es demasiado grande, de manera que en general será mejor una mayor velocidad de avance, siempre que tanto la rapidez de los cálculos internos del robot como la velocidad de giro sean lo suficientemente altas como para permitir un seguimiento adecuado de la trayectoria a elevada velocidad. En cuanto a la velocidad de giro, se observa que al aumentar su valor se obtiene como beneficio un tiempo de recorrido ligeramente menor, además de una recuperación de la trayectoria más rápida después de las curvas, lo cual puede ser de vital importancia en trayectorias con muchos tramos de curvas o con las curvas trazadas de manera muy exacta y cerca de los obstáculos, pero mucho más importante es el efecto de una velocidad de giro muy baja, la cual hace que el robot sea incapaz

de seguir la trayectoria de manera adecuada. Por este motivo, más que buscar la velocidad de giro más elevada posible, lo que se buscará será un valor mínimo que presente un comportamiento suficientemente adecuado, ya que una vez que se tiene un buen seguimiento de la trayectoria, seguir aumentando este parámetro no reporta grandes beneficios.

Por último, se ha estudiado la distancia de "lookahead", que es el parámetro principalmente responsable de la precisión con la que se sigue la trayectoria. Se llega a la conclusión de que al fijar valores elevados, el recorrido efectuado por el robot sigue de manera mucho menos estrecha la trayectoria, pero el tiempo de trayecto disminuye, de manera que dependiendo de la precisión requerida en el seguimiento de la trayectoria, el tiempo que se desee emplear en el trayecto y la cercanía y peligrosidad que presenten los obstáculos, para cada caso se deberá buscar un valor diferente que presentará el comportamiento óptimo según lo esperado. Por otro lado, una distancia de "lookahead" muy baja hará que el robot siga la trayectoria de manera mucho más estricta, por lo que podría ser conveniente en casos en los que sea requerida una enorme precisión, pero la disminución de la distancia de "lookahead" no presenta solamente ventajas, ya que si el valor seleccionado es demasiado bajo al pasar por una curva el robot comenzará a oscilar alrededor de la trayectoria, perdiendo tiempo y energía innecesariamente y pudiendo incluso llegar a perder la trayectoria. Así, tras la experimentación se llega a la misma conclusión que ya se había predicho teóricamente, y es que no existe un valor ideal para el parámetro de distancia de "lookahead", sino que se trata de llegar a un compromiso entre la precisión en el seguimiento de la trayectoria y la rapidez del recorrido, teniendo en cuenta el robot con el que se va a trabajar, la situación que éste se va a encontrar, el comportamiento deseado y los valores de los demás parámetros.

4.1. Conclusión personal

En conclusión, se ha desarrollado un algoritmo de seguimiento de trayectorias basado en el método Pure Pursuit y se ha implementado en un robot Arduino. Se ha experimentado hasta ajustar los parámetros que rigen el movimiento del robot, y finalmente se ha conseguido un comportamiento adecuado que cumple los requisitos deseados. Con el, el robot se mueve de forma mucho más fluida por la plataforma y alcanza su destino de manera bastante precisa.

Por otro lado se han llevado a cabo simulaciones en un software y se ha analizado la influencia de cada uno de los parámetros que rigen el movimiento de un robot al utilizar un algoritmo Pure Pursuit, a fin de tener una idea previa de los valores más adecuados que deberán tener estos parámetros a la hora de implementar un control de este tipo en otro sistema en el futuro.

A partir de aquí, se pueden desarrollar futuros proyectos con la plataforma de robots utilizando este controlador y esta información, desde métodos diferentes de planificación de trayectorias usando este controlador en el robot para la experimentación, hasta la implementación de controladores más precisos y con aplicaciones reales en el mercado utilizando este control Pure Pursuit como base, así como tomar este proyecto como referencia a la hora de intentar implementar un control Pure Pursuit en sistemas diferentes.

Capítulo 5

Bibliografía

[1] Emanuele Vitolo. Multi-robot platform for path planning and control using high-level specifications: Implementation and experiments. Trabajo fin de máster, University of Salerno, 2016.

[2] Gregory Dudek; Michael Jenkin. *Computational principles of mobile robotics*. Cambridge University Press, 2ed. edition, 2010.

[3] MathWorks Inc. https://es.mathworks.com/products/matlab.html, 2017.

[4] Martin Lundgren. Path tracking for a miniature robot. Trabajo fin de máster, Umeå University, 2003.

[5] José Ángel Garrrido Sarasol. Diseño e implementación de un vehículo autónomo terrestre controlado mediante interfaz inalámbrica androidarduino. Trabajo fin de máster, Universidad Politécnica de Valencia, 2016.

[6] José Enrique Escobar Abansés. Estudio del interfaz matlab-ros para el desarrollo de aplicaciones robóticas. Trabajo fin de grado, Universidad de Alcalá, 2016.

[7] Ángel Rivas Toribio. http://studylib.es/doc/196456/los-robots-móviles-o-vehículos-autoguiados- 2017.

[8] Vicent Girbés Juan. Generación de trayectorias de curvatura continua para el seguimiento de líneas basado en visión artificial. Trabajo fin de máster, Universidad Politécnica de Valencia, 2010.

[9] Ronald C. Arkin. *Behaviour-Based Robotics*. The MIT Press, Cambridge, 1998.

[10] G W. Lucas. *A Tutorial and Elementary Trajectory Model for the Differential Steering System of Robot Wheel Actuators*. The Rossum Project, 2001.

[11] R C. Coulter. *Implementation of the Pure Pursuit Path Tracking Algorithm.* Robotics Institute, Carnegie Mellon University, 1992.

[12] Matthew J. Barton. Controller development and implementation for path planning and following in an autonomous urban vehicle. Thesis, 2001.

[13] B. Siciliano and O. Khatib Eds. *Handbook of Robotics.* Springer, 2008.

[14] Antonio Barrientos. *Mecatrónoca, control y automatización.* Editorial MCGRAW-HILL.

Lista de Figuras

Anexos

Anexos A

Experimentación

Aquí se pueden encontrar los resultados de comparar el control PI y el control Pure Pursuit en otras situaciones.

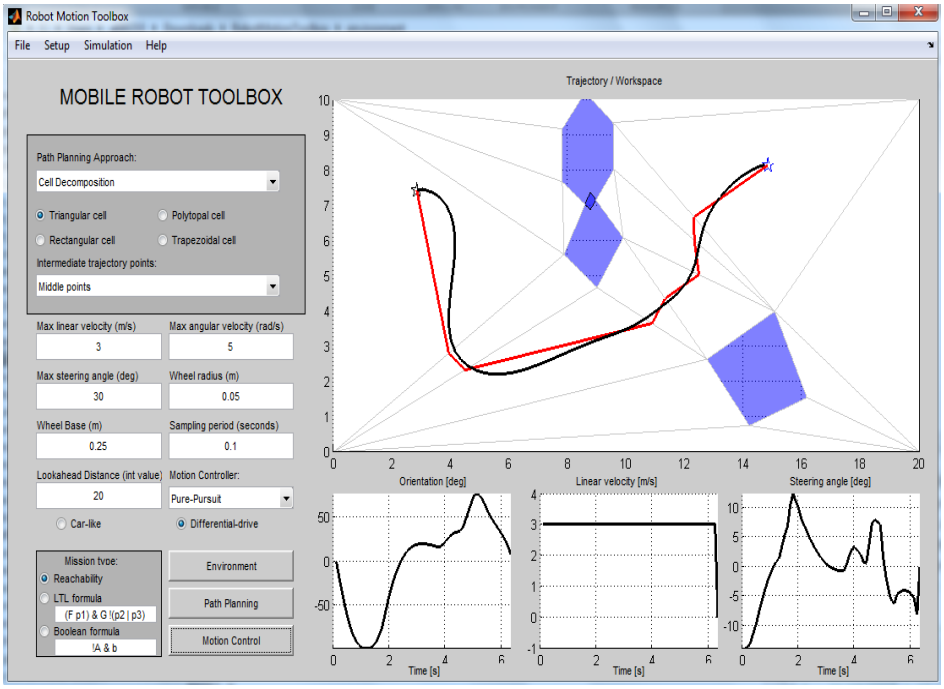

Figura A.1: Situación 2, control Pure Pursuit v=3

En la figura A.1 se observa claramente que el robot no sigue de manera precisa la trayectoria, pero presenta un comportamiento adecuado, ya que no colisiona con ningún obstáculo ni se desvia demasiado de su rumbo. A pesar de ello, como ya se ha comentado, dependiendo del sistema en el que se vaya a implementar podría ser necesaria mayor precisión, lo que requeriría un mayor ajuste de parámetros o el uso de otro controlador.

El comportamiento del robot simulado con el control PI a velocidad de 1 m/s es adecuado, e igualmente válido que al utilizar el controlador Pure Pursuit, pero

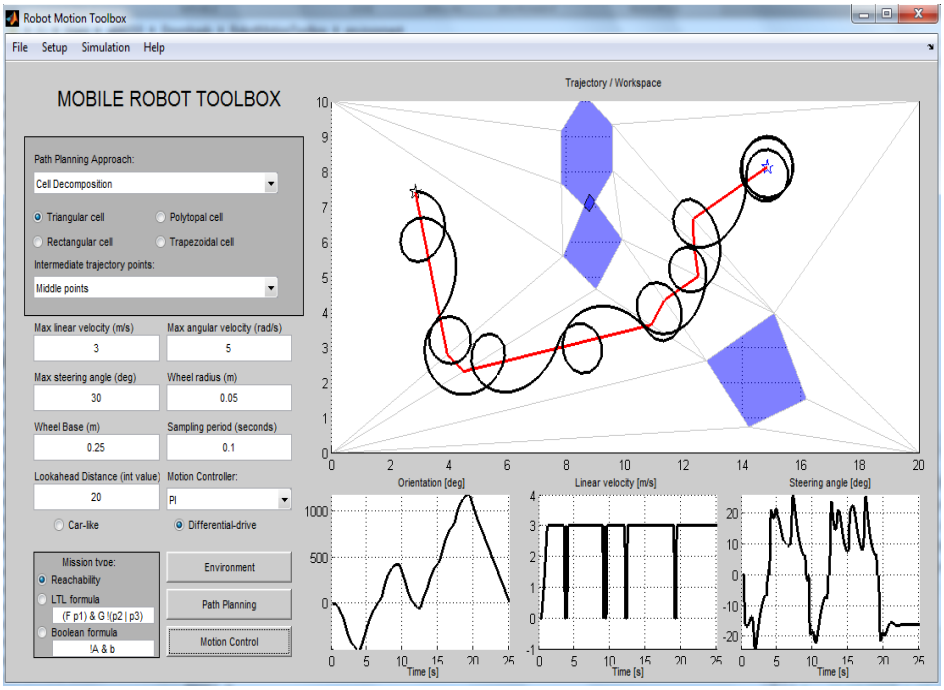

Figura A.2: Situación 2, control PI v=3

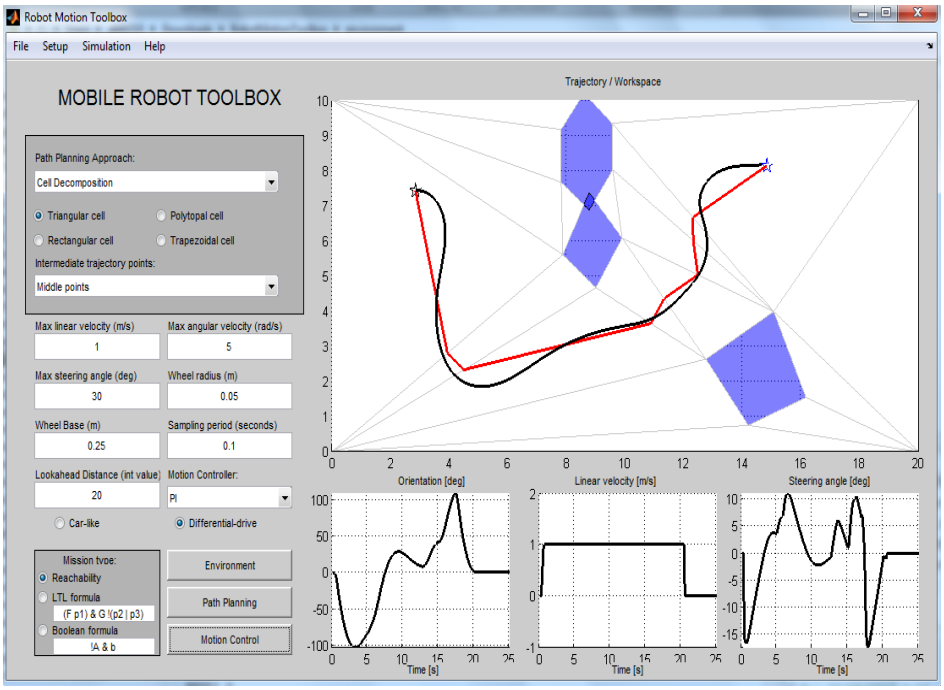

Figura A.3: Situación 2, control PI v=1

presenta una importante desventaja, el tiempo de recorrido es mucho mayor ya que avanaza a menor velocidad, por lo que el controlador Pure Pursuit es más adecuado. El comportamiento del robot con el control PI y una velocidad de 3 m/s no es aceptable en absoluto, ya que se separa demasiado de la trayectoria, oscila, da rodeos y no llega de manera precisa al destino. (Ver figuras A.2 y A.3)

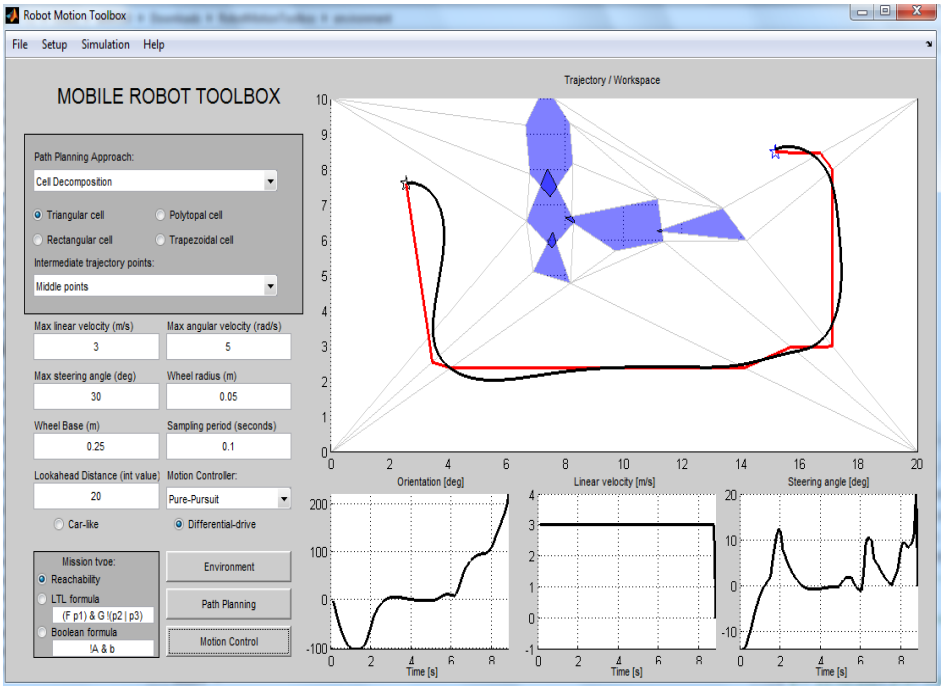

Figura A.4: Situación 3, control Pure Pursuit v=3

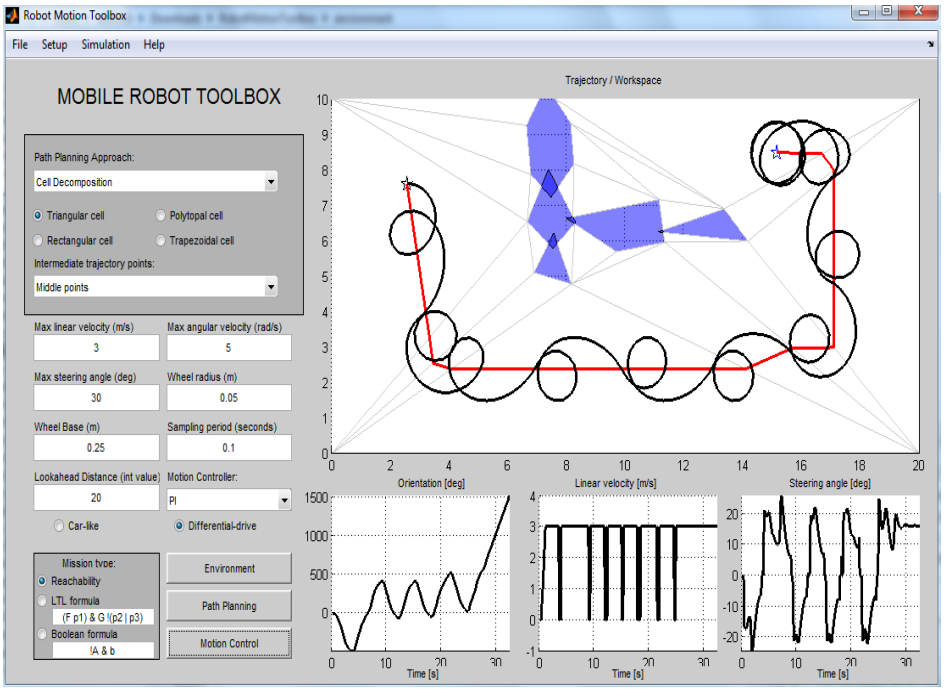

Figura A.5: Situación 3, control PI v=3

En esta tercera situación se puede ver que a pesar de no tener una gran precisión el control Pure Pursuit consigue seguir la trayectoria de manera bastante estricta, teniendo un comportamiento muy cercano al óptimo. En cambio, utilizando el control PI a la misma velocidad que el Pure Pursuit, 3 m/s, el comportamiento del robot sigue sin ser válido en absoluto como los casos anteriores, pero al disminuir la velocidad

a 1 m/s, a pesar de que no presenta oscilaciones demasiado grandes, se observa que el robot no consigue llegar al punto de destino, de manera que su comportamiento tamoco es válido. (Ver figuras A.4, A.5 y A.6)

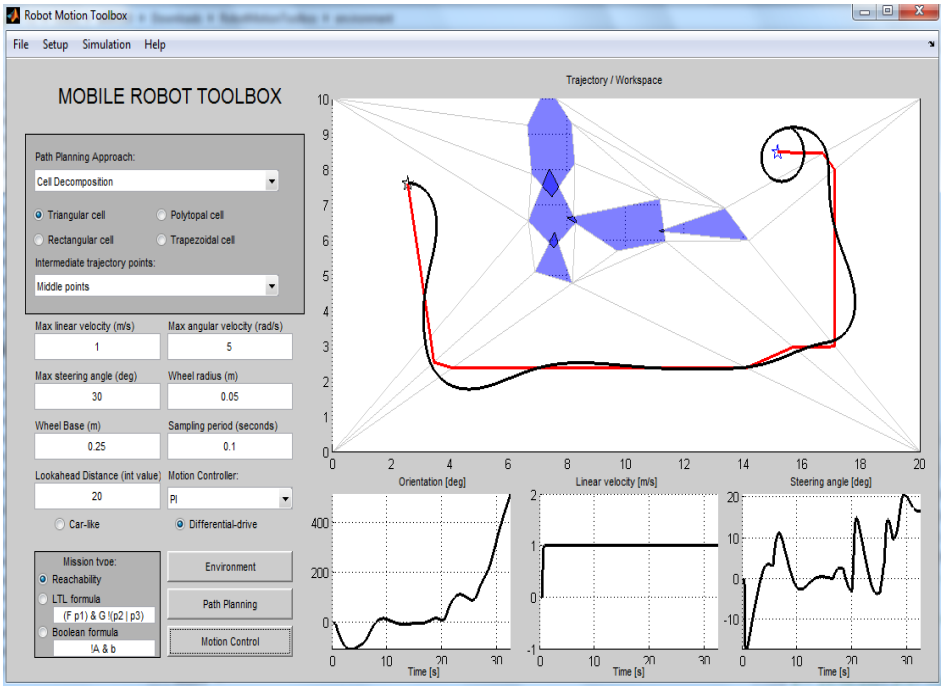

Figura A.6: Situación 3, control PI v=1

Anexos B

Código en C

A continuación se muestra el código en C implementado en el robot para que se comunique con el ordenador del sistema por wifi y siga la trayectoria utilizando el control del tipo Pure Pursuit desarrollado.

```
#include <StandardCplusplus.h>
#include <vector>

using namespace std;

/* INIZIO DICHIARAZIONE VARIABILI */

// VARIABILI COSTITUENTI IL SINGOLO PUNTO E L'INTERO PERCORSO DEL COCHE
vector<vector<float>> path;
vector<float> point;


// VARIABILE COSTITUENTE IL COCHE!
int I_AM = 1;

//Standard PWM DC control
int E1 = 5;      //M1 Speed Control
int E2 = 6;      //M2 Speed Control
int M1 = 4;     //M1 Direction Control
int M2 = 7;     //M1 Direction Control

//Parametri stimatore e ControllerPI
 // Stimatore
 // ControllerPI
int T;
int Ultimotiempo,Nuevotiempo;
int veld,vels;
int r = 6; // diametro ruota in cm
int L = 16; // distanza tra le ruote in cm
const float Kri=1.21;
const float Kle=1;
```

```
const float K=1;//.2;
float roh, roh_old;
float threshold = 4;
int thetaact_degree;


//Parametri ricevuti da WiFi
int num_coche;
float xact;
float yact;
float xact_old;
float yact_old;
float xdes;
float ydes;
float thetaact; // = 2.35; //135 gradi
float thetades;


//Parametri di controllo
bool controller_on = false; // sono arrivato o no? sta nel main. Si...
...attiva quando leggo un nuovo path e si disattiva quando vi arrivo

int mem_index = 0; //indice corrente da cui prelevare il punto i-esimo...
...dal vettore "path"

bool ascolto_pacchetto = true; // serve per dire al controller se...
...ascoltare il frame-packet o usare lo stimatore

bool sono_partito = false; // serve per far orientare SOLO ALL'INIZIO,...
...qualora ce ne fosse bisogno, il robot

bool prima_iter_fatta = false; // faccio un controllo per capire se...
...man mano invece che avvicinarmi mi sto allontanando;
                                // in tal caso fermo tutto; per capirlo...
  ...compare il nuovo valore di roh col vecchio.
                                // Questa variabile serve per capire...
  ...se sono alla prima iterazione, perche in tal caso, assegno
                                // al vecchio valore di roh il valore...
  ...attuale di roh (non ho un istante precedente); LO DEVO FARE
                                // perche senno roh_old (che sara 0 non...
 ...essendo stato assegnato) sara gia piu piccolo di roh attuale!

// questi due servono per capire dove deve girare il robot per orientarsi
bool check = false;
bool check2 = false;

bool prima_stima = true;
```

```
/* FINE DICHIARAZIONE VARIABILI */


void listen_packet_WiFi(){
  if(Serial1.available()){
      char key = 'x';
      if(key == 'x'){
        String temp = "";
        while(key == 'x'){
            char c = Serial1.read();
            delay(10);
            if(c == 'y'){
                key = c;
              }
            else{
                temp+=c;
              }
        }
        xact = atof(temp.c_str());
        Serial.println("xact");
        Serial.println(xact);
      }
      if(key == 'y'){
        String temp = "";
        while(key == 'y'){
            char c = Serial1.read();
            delay(10);
            if(c == 'T'){
                key = c;
              }
            else{
                temp+=c;
              }
        }
        yact = atof(temp.c_str());
        Serial.println("yact");
        Serial.println(yact);
      }
      if(key == 'T'){
        String temp = "";
        while(Serial1.available()){
            char c = Serial1.read();
            delay(10);
            temp+=c;
        }
        thetaact = atof(temp.c_str());
        Serial.println("thetaact");
```

```
        Serial.println(thetaact);
    }

    xact = xact*100;
    yact = yact*100;
  }
  //delay(50);
}



void listen_path_WiFi(){
  // Se ricevo un nuovo path, riabilito il controller
  while(!(Serial1.available())){;}
  if(Serial1.available()){
      char key = Serial1.read();
      // se il path e vuoto perche magari sono gia nella posizione...
      ...desiderata...non fare niente
      if(key == 'F')
        ;
      else{
        num_coche = I_AM;

        while(key != 'F'){
        if(key == 'x'){
          String temp = "";
          while(key == 'x'){
            while(!(Serial1.available())){;}
            char c = Serial1.read();
            delay(10);
            if(c == 'y'){
                key = c;
              }
            else{
                temp+=c;
              }
              delay(10);
          }
          point.push_back(atof(temp.c_str()));
        }
        if(key == 'y'){
          String temp = "";
          while(key == 'y'){
            while(!(Serial1.available())){;}
            char c = Serial1.read();
            delay(10);
            if(c == 'x'){
                key = c;
```

```
            }
            else if(c == 'F'){
                key = c;
            }
            else{
                temp+=c;
            }
            delay(10);
          }
          point.push_back(atof(temp.c_str()));
        }
        path.push_back(point);
        point.clear();
        delay(10);
      }
      }
  }
  analogWrite (E1,98);
  digitalWrite(M1,HIGH);
  analogWrite (E2,100);
  digitalWrite(M2,HIGH);
  delay(50);
  analogWrite (E1,0);
  digitalWrite(M1,LOW);
  analogWrite (E2,0);
  digitalWrite(M2,LOW);
  //delay(50);

  // trasformo in cm
  for(int m = 0; m < path.size(); m++){
      path[m][0] = path[m][0] * 100;
      path[m][1] = path[m][1] * 100;
    }

//  for(int m = 0; m < path.size(); m++){
//      Serial.print("x: ");
//      Serial.println(path[m][0]);
//      Serial.print("y: ");
//      Serial.println(path[m][1]);
//      delay(50);
//    }
}


void estimationmodel(){

    if(prima_stima == true){
      prima_stima = false;
```

```
    Nuevotiempo=millis();
    T = 0;
    Ultimotiempo=Nuevotiempo;
  }
  else{
    Nuevotiempo=millis();
    T=Nuevotiempo-Ultimotiempo;
  }

  float ur=veld;
  float ul=vels;
  ur=veld*0.05;
  ul=vels*0.05;
  thetaact=thetaact-(T*r*(ur-ul)/L*0.001);//L = 2*R
  thetaact_degree =  (thetaact * 180)/3.14;


  xact=xact+((T*4*(ul+ur)*cos(thetaact))*0.001/2);
  yact=yact+((T*4*(ul+ur)*sin(thetaact))*0.001/2);
  Ultimotiempo=Nuevotiempo;

//    Serial.println("xact");
//    Serial.println(xact);
//    Serial.println("yact");
//    Serial.println(yact);
//    Serial.println("thetaact");
//    Serial.println(thetaact);
//    Serial.println("thetaact_DEEGREE");
//    Serial.println(thetaact_degree);
}

void estimationmodel_rot(int rot_degree){
  float th = 0;
  int th_deg = 0;

  if(prima_stima == true){
    prima_stima = false;
    Nuevotiempo=millis();
    T = 0;
    Ultimotiempo=Nuevotiempo;

  }
  else{
    Nuevotiempo=millis();
    T=Nuevotiempo-Ultimotiempo;
  }
```

```
if(check){
  if(check2 == true){ // antiorario
    while(th_deg < rot_degree){
        Nuevotiempo=millis();
        analogWrite (E1,110);
        digitalWrite(M1,HIGH);
        analogWrite (E2,0);
        digitalWrite(M2,LOW);
        delay(50);
        float ur=50*0.05;
        float ul=-50*0.05;
        T=Nuevotiempo-Ultimotiempo;
        th = th +(T*r*(ur-ul)/L*0.001);//L = 2*R
        th_deg =  (th * 180)/3.14;
        Ultimotiempo=Nuevotiempo;
    }
  }
  else{ //orario
    while(th_deg < rot_degree){
        Nuevotiempo=millis();
        analogWrite (E1,0);
        digitalWrite(M1,LOW);
        analogWrite (E2,100);
        digitalWrite(M2,HIGH);
        delay(50);
        float ur=-50*0.05;
        float ul=50*0.05;
        T=Nuevotiempo-Ultimotiempo;
        th =th +(T*r*(ul-ur)/L*0.001);//L = 2*R
        th_deg =  (th * 180)/3.14;
        Ultimotiempo=Nuevotiempo;
    }
  }
}
else{
  if(check2 == true){ // orario
    while(th_deg < rot_degree){
        Nuevotiempo=millis();
        analogWrite (E1,0);
        digitalWrite(M1,LOW);
        analogWrite (E2,100);
        digitalWrite(M2,HIGH);
        delay(50);
        float ur=-50*0.05;
        float ul=+50*0.05;
        T=Nuevotiempo-Ultimotiempo;
        th =th +(T*r*(ul-ur)/L*0.001);//L = 2*R
        th_deg =  (th * 180)/3.14;
```

```
            Ultimotiempo=Nuevotiempo;
          }
        }
        else{ //antiorario
          while(th_deg < rot_degree){
            Nuevotiempo=millis();
            analogWrite (E1,110);
            digitalWrite(M1,HIGH);
            analogWrite (E2,0);
            digitalWrite(M2,LOW);
            delay(50);
            float ur=50*0.05;
            float ul=-50*0.05;
            T=Nuevotiempo-Ultimotiempo;
            th =th +(T*r*(ur-ul)/L*0.001);//L = 2*R
            th_deg =  (th * 180)/3.14;
            Ultimotiempo=Nuevotiempo;
          }
        }
      }
          Serial.println("FATTA ROTAZIONE");

      thetaact = thetades;
      analogWrite (E1,0);
      digitalWrite(M1,LOW);
      analogWrite (E2,0);
      digitalWrite(M2,LOW);
      delay(100);

}

void ControlPI(){

      roh = sqrt((((xdes) - (xact)) * ((xdes) - (xact))) +...
     ... (((ydes) - (yact)) * ((ydes) - (yact))));  //  e(KT)

//      Serial.println("thetaact in rad");
//      Serial.println(thetaact);
//      Serial.println("thetades in rad:");
//      Serial.println(thetades);
      Serial.println("distanza pitagorica:");
      Serial.println(roh);


      int large=large+(int(roh)*K); // azione integrale   u(t) = Kp*e(t) +
                              ...Ki*(e(t) cumulato nel tempo) ---->
                                      Kp e fittizio (=1) / e(t) sarebbe
                                      roh / Ki e fittizio (=1)
```

```
    // Il controllo qui in basso e effettuato in quanto viene
    // tenuto conto della PERIODICITA' di atan2
    float diff_ang = (thetades-thetaact);
    if(diff_ang > 3.14)
      diff_ang = 2*3.14 - diff_ang;
    else if (diff_ang < - 3.14)
      diff_ang = -2*3.14 - diff_ang;

    float gir=0;

    if(check){
      if(check2)
        ;
      else
        gir = -gir;
    }
    else{
      if(check2)
        gir = -gir;
      else
        ;
    }


//    Serial.println("large:");
//    Serial.println(large);
//    Serial.println("gir:");
//    Serial.println(gir);
    veld=255*int(large>255)+large*int(large>0)*int(large<255)+(int)...
    ...(gir*8)*int(roh>threshold);
    vels=255*int(large>255)+large*int(large>0)*int(large<255)-(int)...
    ...(gir*8)*int(roh>threshold);

    //garantisco il minimo contributo PWM per far muovere i motori
    if(vels < 49 && veld < 49){
        vels = vels + 62;
        veld = veld + 62;
    }

//    Serial.println("veld:");
//    Serial.println(veld);
//    Serial.println("vels:");
//    Serial.println(vels);
}
```

```
bool first_time = true;

void controller_final(){


while(mem_index!= path.size()){

    xdes = path[mem_index][0];
    ydes = path[mem_index][1];


    Serial.println("x_des");
    Serial.println(xdes);
    Serial.println("y_des");
    Serial.println(ydes);
    ////////
    /////// INIZIO ALGORITMO CONOSCENZA VERSO DI ROTAZIONE
    ////////

    // variabili booleane per conoscere il verso di rotazione piu conven
    check = false;
    check2 = false;

    // calcolo l'orientamento per giungere al punto
    thetades = atan2(ydes - yact, xdes - xact);


    //int thetaact_degree =  (thetaact * 180)/3.14; // conversione in gra
    thetaact_degree =  (thetaact * 180)/3.14; // conversione in gradi

    if (thetaact_degree < 0.0) {
      thetaact_degree += 360.0; // normalizzazione
    }

    int thetades_degree =  (thetades * 180)/3.14; // conversione in gradi

    if (thetades_degree < 0.0) {
      thetades_degree += 360.0; // normalizzazione
    }

    int diff_degree;

    if(thetades_degree >= thetaact_degree){
      diff_degree = thetades_degree - thetaact_degree; // differenza in g
      check = true;
      if(diff_degree <= 180)
```

```
    check2 = true;
  else{
    check2 = false;
    diff_degree = 360 - diff_degree;
  }
}
else{
  diff_degree = thetaact_degree - thetades_degree; // differenza in g
  check = false;
  if(diff_degree <= 180)
    check2 = true;
  else{
    check2 = false;
    diff_degree = 360 - diff_degree;
  }
}

////////
/////// FINE ALGORITMO CONOSCENZA VERSO DI ROTAZIONE
////////

   if(diff_degree > 5 ){//&& first_time){
      estimationmodel_rot(diff_degree);
      first_time = false;
   }



Serial.println("diffdeggree");
Serial.println(diff_degree);


ControlPI();

if(prima_iter_fatta == false)
    roh_old = roh;

          prima_iter_fatta = true;


while( (roh > 6 && mem_index != (path.size() - 1)) ||...
...(roh > 3 && mem_index == (path.size() - 1)) ...
...&& !( roh > roh_old && (fabs(roh_old - roh)> 0.50)...
...&& prima_iter_fatta == true)){

    analogWrite (E1,70);
    digitalWrite(M1,HIGH);
```

```
        analogWrite (E2,70);
        digitalWrite(M2,HIGH);
        ControlPI();

        estimationmodel();


      if(roh < roh_old && prima_iter_fatta == true)
        roh_old = roh;



    Serial.println("xact");
    Serial.println(xact);
        Serial.println("yact");
    Serial.println(yact);

    Serial.println("veld");
    Serial.println(veld);
    Serial.println("vels");
    Serial.println(vels);
    Serial.println("thetaact");
    Serial.println(thetaact);
        Serial.println("thetades");
    Serial.println(thetades);

        delay(20);
    }
    mem_index++;
    Serial.println("mem_i");
    Serial.println(mem_index);
                  prima_iter_fatta = false;


   }



        analogWrite (E1,0);
        digitalWrite(M1,LOW);
        analogWrite (E2,0);
        digitalWrite(M2,LOW);
 controller_on = false;

}
```

```
void setup() {
  Serial1.begin(9600);
  Serial.begin(9600);
}


bool okk = false;
void loop() {
//     if(ccc == 0){
//       delay(2000);
//       ccc++;
//     }
//     else if(!pp){
//       controller_final();
//     }
//     else
//       ;
//
//         delay(50);

  if(Serial1.available() && controller_on == false && okk == false){
    Serial.println("Ricevuto");
      char key = Serial1.read();
      if(key == 'P'){
        while(!(Serial1.available())){;}
        key = Serial1.read(); // legge la 'c' prima del numero del coche
        while(!(Serial1.available())){;}
        key = Serial1.read(); // ora legge il numero del coche
        if(I_AM == (key - '0')){
          path.clear();
          listen_path_WiFi();
          Serial.println("Ricevuto");
    okk = true;
        }
      }
  }
  if(Serial1.available() && controller_on == false && okk == true){
  while(!(Serial1.available())){;}

        char key;
        key = Serial1.read();
        // pongo il num_coche a 0 cosi se key != c (altro pacchetto)...
        ...non succede nulla (return)
```

```
        // perche num_coche = 0 PER CONVENZIONE NON CORRISPONDE...
        ...A NESSUN COCHE
        num_coche = 0;
        if(key == 'c'){
          String temp = "";
          while(key == 'c'){
            char c = Serial1.read();
            delay(10);
            if(c == 'x'){
              key = c;
            }
            else{
              temp+=c;
            }
          }
          num_coche = atof(temp.c_str());
        }
        // altro controllo di verifica: se il pacchetto e' buono ma e' pe
        if(I_AM == num_coche){
          listen_packet_WiFi(); // leggi pos e orientamento aggiornati
          Serial.println("Ricevuto");
          controller_on = true;
        }
    }



    //viene abilitato solo dopo aver letto il path
    if(controller_on == true){
      okk = false;
      controller_final();
    }
    delay(20);

}
```