İSTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

PATH TRACKING METHODOLOGIES FOR MOBILE ROBOTS

M.Sc. THESIS

Sara HOSSEINI

Department of Control and Automation Engineering

Control and Automation Engineering Program

Thesis Advisor: Prof. Dr. Hakan TEMELTAŞ

JUNE 2016

İSTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

PATH TRACKING METHODOLOGIES FOR MOBILE ROBOTS

M.Sc. THESIS

Sara HOSSEINI
(504121146)

Department of Control and Automation Engineering

Control and Automation Engineering Program

Thesis Advisor: Prof. Dr. Hakan TEMELTAŞ

JUNE 2016

Sara Hosseini, a M.Sc. student of İTU Graduate School of Control and Automation Engineering student ID 504121146, successfully defended the thesis entitled "PATH TRACKING METHODOLOGIES FOR MOBILE ROBOTS", which she prepared after fulfilling the roo0equirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor : **Prof. Dr. Hakan TEMELTAŞ** .....
İstanbul Technical University

Jury Members : **Yrd. Doç. Dr. Sıddık Murat Yeşiloğlu** .....
İstanbul Technical University

Yrd.Doç.Dr. İlker Üstoğlu .....
Y.T.U

Date of Submission : 02 May 2016
Date of Defense : 05 June 2016

To my parents and husband,

FOREWORD

I would like to thank my advisor Prof. Dr. Hakan TEMELTAŞ, for guiding and supporting me. He have set an example of excellence as a researcher, mentor, instructor who helped me remarkably through this long road.

I would specially like to thank to my dear parents who helped me morally and financially through my education and always gave me encouraging applause in every step of my life.

I should thank to my dear spouse for supporting me all the way down up to here and help me with my thesis in any way he could.

June 2016 Sara HOSSEINI

TABLE OF CONTENTS

Page

ABBREVIATIONS

GPS	**:** Global Positioning System
2D	**:** Two Dimension
3D	**:** Three Dimension
ICR	**:** Instantaneous Center of Rotation
DOF	**:** Degree of Freedom
DOS	**:** Degree of Steerability
ICC	**:** Instantaneous Center of Curvature
sptSet	**:** Shortest Path Tree Set
PRM	**:** Probabilistic Road Map
RRT	**:** Rapidly Exploring Random Tree

SYMBOLS

$\{X_R, X_R\}$	: Coordinates of Robot With Respect to the Global Frame
$\{X_L, X_L\}$	: Global Frame
$\theta, \theta_i, \theta_o, \theta_{error}$	: Angular Difference
$\mathbf{R}(\theta)$	: Mapping Matrix
ζ	: Pose (Position and Orientation) of Robot
α	: Orientation of Wheel With Respect to Local Frame Due to Axis *X*
P	: Center of Wheel
$l, d, d`$	: Distance Components
$\{\dot{x}, \dot{y}\}$	: Linear Velocities
$\dot{\theta}$	: Angular Velocity
$\varphi(t)$	: Rotational Posture of Robot
β	: Angle of Wheel Relative to Body of Robot
r	: Radius of Wheel
a_x, a_y	: Unit Vectors
v, v_r, v_l	: Linear Velocity
R	: Radius of ICR
δ_m	: Degree of Mobility
δ_s	: Degree of Steerability
δ_M	: Degree of Maneuverability
(x_r, y_r, θ_r)	: Robot`s Coordinates in Ackerman Drive
(x_w, y_w, θ_w)t	: Robot`s Position in World Coordinates
S_0, G, V	: Nodes
$g(n), f(n), h(n)$	: Costs
E	: Edge Cost
$C(\mathcal{R})$	: Configuration Space
$\mathcal{T}(E)$	: Trapezoidal Map
$\mathcal{C}$	: Free Space
N	: Number of Nodes in PRM
e_0	: Orientation Error
k_t, k_p	: Gains
θ_c	: Current Heading Angles
$\{x_g, y_g\}$	: Position of Look-ahead Distance
$\{x_c, y_c\}$	: Current Position of Robot
γ	: Curvature Arc of Robot
S	: Unit Vector of Screw
S_0	: Moment Vector
$\overline{\overline{S}}, \overline{\overline{S}}_r, \overline{\overline{S}}_t$	: Screws
Δ	: Error
k_{ax}, k_{ay}	: Coordinates on Maint Path
k_{px}, k_{py}	: Coordinates on Surveyed Path

LIST OF FIGURES

Page

PATH TRACKING METHODOLOGIES FOR MOBILE ROBOTS

SUMMARY

In the world of robots, there are several types of mobile robots that are able to work in indoor or outdoor environments. Their structure are different from each other as the type of wheels or their architecture of positioning on the robot differs.
In chapter one, an introduction of robot world and what is done in literature survey is discussed and the development that is done in this plane is introduced.
In chapter two, firstly the position of the robot`s frame with respect to the world frame is defined with its related formulation, then a brief introduction of different kinds of wheels and their behavior is introduced. Later on some different kind of robots are introduced such as differential drive robots, Ackerman drives, omni directional drives and so on.
In chapter three, path planning methods are introduced. These methods helps the robot to find the most optimize and short path to reach its final destination from its current initial point. Using a suitable method and the most optimized one is a main factor in path planning. The more complicated the path planning is, much time is used for the robot to reach its destination and therefore it will be a time consuming procedure and the percentage of battery usage will increase. To overcome these drawbacks one should think of a logical path planning algorithm to implement to the robot and lead it in a smooth and time saving way to the destination without colliding any obstacles or walls in its way. So some algorithms from the most simple and old one to the most recent and optimized ones are introduces and at the last part the path planning that is used in this thesis will be presented. First Dijkstra`s and A* methods are introduced. A* used Dijkstra`s method and uses weighted graphs to help robot through the most efficient path. It uses the movement costs $g(n)$ and $h(n)$ which are the cost from the current position of the robot to the initial point and the cost from the current position of the robot to the goal point respectively. Summing up these costs leads to cost function $f(n)$ which determines the most efficient node to select among the cells which are around robot`s current location. The less the cost is the more priority it has in A* method. The previous method was good in known environments. D* method uses heuristic method in partially known or unknown environments in path planning and uses A* method backwards from destination to the current point in each step to find an obstacle free or no collision with walls, way. After that visibility graph method is introduced which uses Trapezoidal map to find an optimum short path using visible nodes who see each other. In this method the shape of robot and obstacle are also considered as polygons. If we consider our robot as a point robot then the robot can move freely in our space by defining the shortest path and will never collide the

obstacles. However, if our robot is a polygon robot, and we have considered a point in the robot as a reference point, then there is a change that the robot collides with obstacles in this path. So due to the dimensions of the robot, an inflation should be implement to the walls and obstacles to avoid the collision. Afterwards, cell decomposition method introduces a method that separates the whole map into cells and uses those cells to find a suitable road map for the mobile. At the end the probabilistic road map method which is used in this thesis is introduced. This method uses probabilistic node selection in configuration space and by connecting these nodes together finds a bunch of roadmaps for the robot to follow. The number of nodes can be selective. This method is less complicated with respect to other methods so it can be used in much complex environments.

In chapter four, the robot uses path planning method introduced in the previous chapter for path tracking and reach its destination. Different kinds of methods have been introduced namely; follow-the-carrot, pure pursuit and vector pursuit methods. Follow the carrot method finds the coordination of final point and points directly toward it. This method is a simple method in literature and it is not so practical because it follows the look ahead distances directly with a straight line, without considering the trajectory to be followed. Pure pursuit method, however, uses look ahead distance for following the trajectory. It computes the arc necessary for the robot to find its ways and come back to the road. This values depends on two elements, first one, the x position of the goal point and the second one distance from center of robot to the goal point , L.These two methods does not consider the orientation of the robot at the goal point, however the third method, vector pursuit, obeys from pure pursuit method but also considers the orientation of the look-ahead distance of robot. Vector pursuit uses screw theory which was introduced by Sir Robert S.Ball in 1900, the method is developed for applications in kinematics and statics of rigid body and provides a mathematical formulation for the geometry of lines which are central to rigid body dynamics. Any motion of the rigid body can be described as a rotation about a line in space with an associated pitch.

In chapter five, simulations have done in MATLAB using path tracking method, a road map has been constructed, later on the robot pursuits that path using MATLAB codes and find destination. The linear velocity in path tracking has an important impact on the behavior of the robot in sharp corners. As speed increases the overshoot in corners also increase and robot has a less accuracy in sharp corners. However, for the low speeds, it can be seen that the robot pursuits the corners in a more accurate way. Using PRM method and finding desired roadmap, the main problem now is to pursue this road in a most perfect way. As mentioned the impact of linear velocity and look ahead distance has a major role in path tracking. Two scenarios are discussed here.

The first scenario is considering a constant look ahead distance of 0.5 meter and changing the linear velocity of the robot (0.5, 1, 1.5 and 2 m/s) to see the difference between them and approach the diagram of error between these velocities for both pure pursuit and vector pursuit methods. The standard deviation error of these four velocities for two methods of pure pursuit and vector pursuit are depicted. The results Show that by increasing velocity of the robot for a constant look-ahead distance the standard deviation error for he pure pursuit increases, and this is a flaw for this method because imagining a more increased velocity for the robot as coming down the hill and gaining velocities more than 2 m/s the results of the error will be very dramatic and un unneglectable. On the other hand, for vector pursuit method Standard deviation error is almost constant all over the road and much stability can be seen during following the path.

The second scenario in considering a constant velocity of 1.5 m/s and changing look-ahead distances as 0.5, 1, 1.3 and 1.7 meters. Sketching standard deviation error for pure pursuit and vector pursuit methods for this scenario also results that for vector pursuit method standard deviation error remains almost constant all over the road, on the other hand large overshoots and errors appear by increasing the look-ahead distance.

In chapter 6, conclusion is made of what is done in this thesis and what can be done in future studies is mentioned. The simulink results shows that the pure pursuit method Works weaker than vector pursuit method in path tracking methodologies, and results in higher errors but vector pursuit method is much smoother and the accuracy of following the path for this method is higher. Despite the complexity of mathematical calculations of the vector pursuit, this method is preferred to other ones.

MOBİL ROBOTLAR İÇİN ÇİZGİ İZLEYEN (YOL TAKİP EDEN) METODOLOJİLER

ÖZET

Mobil robotlar dünyasında, iç ve dış ortamda çalışabilen fazla sayıda robot çeşidi bulunmaktadır ve bu robotların yapım sistemi, tekerleklerin yapım şekli ve robot üzerinde konumu birbirinden farklı olmaktadır. Bu farklılıklar, robotların çalışma şeklinde önemli rol oynamaktadır. Tekerleklerin formu ve yapısı, robotların özgürlük derecesinin artmasına sebep olur ve özgürlük derecesi arttıkça, daha kesin ve verimli yol takip metotları çıkartmaya yardımcı olur. Ayrıca, robotların yapım şekli, örneğin tekerlek adeti, tekerlek tipi, yönlendirme kabiliyeti ve manevra yeteneği robotların davranışında çeşitli rol oynamaktadır. Dolayısıyla, çeşitli robot yapı şekilleri tanıtılacak ve tez konusu olan robot yapısını sonraki bölümlerde tanıyacağız.

Birinci bölümde, robotlar dünyasına bir tanıtım gerçekleştiriyoruz ve bu konuda literatür taraması yapacağız ve sonra yapılan geliştirmeleri mercek altına alacağız.

İkinci bölümde, ilk matematiksel olarak robot çerçevesi ve dünya çerçevesi arasında olan ilişkiyi anlatacağız, sonra çeşitli tekerlek yapıları ve davranışlarını tanıyacağız. Daha sonra, farklı robot yapım sistemlerini göreceğiz. Bu sistemlerden bazıları: diferansiyel tahrik robot, Ackerman tahrik robot ve Omni yönlü tahrik robot vs olmaktadır. Araştırmamızda kullanacağımız robot tipi, çift düzenlenebilir standart tekerlekli (Two Steered Standard Wheels) standart diferansiyel robot olmaktadır.

Üçüncü bölümde, yol planlaması metotları tanıtılacaktır. Planlama metotları, robotu en kısa yoldan ve en hızlı şekilde başlangıç noktasından varış noktasına ulaştırmaya yardımcı olacaktır. Robotun izleyeceği yol haritası ne kadar karmaşık ise, robotun doğru yolu bulması ve varış noktasına varması daha fazla zaman alıcı olup ve neticede daha fazla batarya (pil) kullanımına yol açmaktadır. Bu engeli aşmak için, daha mantıklı ve verimli yol planlama algoritması oluşturmak gerek. Robota daha kesin bir yol planı uyguladığımızda, daha akıcı ve daha az zaman harcayarak hiçbir duvar veya engele rastlamayacak şekilde varış noktasına varacaktır. Dolayısıyla, en basit, en eski, en verimli ve en yeni algoritmalardan oluşan çeşitli yol planlama algoritmalarını tanıtıp ve sonunda bu araştırmada kullandığımız yol planlama metodunu tanıyacağız. İlk olarak Dijkstra's ve A* metotları tanıtılacaktır. A* metodu, Djikstra's ve Ağırlıklı Graf metotlarını kullanarak robot için en verimli yolun bulunmasına yardım etmektedir. Bu metot, hareket maliyeti olan $g(n)$ ve $h(n)$ parametrelerini kullanmaktadır. Robotun şimdiki konumundan başlangıç noktasına kadar olan maliyet $g(n)$ ve $h(n)$ şimdiki konumundan varış noktasına olan maliyet olmaktadır. Bu iki

maliyeti topladığımızda, maliyet fonksiyonu olan $f(n)$ elde edilir. $f(n)$ robotun şimdiki konumunun etrafında olan hücrelerden en verimli hücreyi belirler. Her ne kadar maliyet az olursa, A* metodunda daha öncelikli olmaktadır. Bahsi geçen metot bilinen çevrelerde kullanılmak için çok iyi bir metot. D* metodu, buluşsal metodunu kullanarak, kısmen bilinen çevrelerde yol planlamasına yardımcı oluyor ve A* metodunu kullanarak, başlangıç noktasına geri dönmek için engelsiz ve çarpışmasız bir yol bulmak için kullanılıyor. Sonra, Görünürlük Graf metodunu tanıyacağız. Bu metot 'da ikizkenar haritası kullanarak, birbirini tanımlayabilen görünebilir noktalar yardımı ile optimum ve en kısa yol planlaması yapılıyor. Daha sonra, hücre bozuşma metodunu tanıyacağız. Bu metot 'ta robotun yapısı ve etrafındaki engeller poligon şeklinde düşünülür. Eğer robotu bir nokta farz edersek, en kısa bulduğunda hiçbir engele takılmadan rahatça çevrede dolaşabilir ve varış noktasına varır. Ama eğer robotu poligon şeklinde düşünüp ve robot üzerinde bir noktayı referans nokta olarak farz edersek, durum değişir ve robot yolu bularken engellere çarpabilir. Dolayısıyla, robot ölçülerini baz alarak, çarpmayı önlemek için engeller ve duvarlara enflasyon (şişme) uygulamak zorundayız. Daha sonra hücre dağılma metodunu tanıyacağız. Bu metot'ta tüm harita hücrelere bölünüyor ve her hücre en verimli ve kısa yolun bulunmasında mobil robota yardımcı oluyor. En son, bu araştırmada kullanılan olasılıklı yol haritası metodunu tanıyacağız. Bu metot, yapılandırma uzayında, olasılık üzere noktaları seçip ve birbirine bağlayarak robotun takip etmesi için birkaç yol belirler. Noktaların sayısı belirlenebilir. Bu metot diğer metotlara nazaran daha basit ve karmaşık çevrelerde kullanılabilir bir metot olmaktadır.

Dördüncü bölümde, önceki bölümde tanıtılan yol planlama metotları en verimli yolu bulmak ve varış noktasına ulaşmak için robot üzerinde uygulanıyor. Çeşitli metotlar tanıtıldı; follow-the-carrot, Saf Takip, ve Vektör Takip metotları. Follow-the-carrot metodu, varış noktası ve ona varan noktaların koordinatlarını buluyor. Bu metot literatür 'de basit olarak biliniyor ve çok kullanışlı bir metot olmamaktadır. Bunun nedeni ise, direk olarak robotun önünde olan mesafeyi ölçerek yol planını çıkartması ve robotun takip edeceği yörüngeyi kayde almaması. Ama Saf Takip metodu, önündeki mesafeyi yolu takip etmek için kullanıyor. Bu metot'ta, robotun yolunu bulması için gerekli yay hesaplanıyor ve robot tekrar yola dönmek için bu yayı kullanıyor. Bu değerler, iki değişken ile belirlenir, birincisi varış noktasının konumu olan (x) ve diğeri robotun merkezinden varış noktasına olan (L) mesafesi. Bu iki metot, robotun yönünü varış noktasında kayda almıyorlar, hal bu ki üçüncü metot yani Vektör Takip metodu, saf takip yöntemini kullanmasının yanında, robotun yönünü de varış noktasında ayarlıyor. Vektör takip metodu, vida teorisini kullanmaktadır. Vida teorisi, 1900 yılında Sir Robert S. Ball tarafından öne sürülmüş ve katı gövde dinamiğin'de katı gövdenin merkezinde olan çizgilerin geometrisi için, kinematik ve statik olarak

matematiksel formüller üretmiştir. Katı gövdenin her hareketi, merkezinden çıkan çizgilerin rotasyonu üzerinden anlatılabilir.

Beşinci bölümde, MATLAB ortamında çeşitli metotların simülasyonları yapılıyor. Yol haritası çıkartılıyor ve MATLAB ortamına uygulanarak robotun yolu izlemesi ve varış noktasına varması simüle ediliyor. Lineer hız, keskin virajlarda robotun davranışlarında çok önemli rol oynamaktadır. Hız yükseltikçe, virajda aşım da yükseliyor ve robot virajda kesinliğini kaybediyor. Ama, düşük hızlarda, robot virajları daha kesin bir şekilde takip ettiğini görüyoruz. PRM metodunu kullanarak istenilen yol haritasını bulduktan sonra, en önemli nokta bulunan yolu en iyi şekilde nasıl takip edilmesidir. Önceden bahsedildiği gibi, lineer hız ve öndeki mesafe, yol takibi işleminde en önemli iki faktör olmaktadırlar. İki senaryo hakkında bahsedilecektir;

Birinci senaryoda, robotun öndeki mesafesini 0.5m sabitleyip ve lineer hızı değiştirerek (0.5, 1, 1.5 ve 2m/s) çeşitli hızlarda robotun davranışlarını ve hata diyagramını, Saf Takip ve Vektör Takip Metotlarında incelenecektir. Her iki metotta, belirlediğimiz dört hız için Standart sapma hatası gösterilecektir. Sonuçlara baktığımızda, sabit ön mesafede, hız yükseldikçe Saf Takip metodunun standart sapma hatası da yükseliyor ve bu olay bu metot için bir dezavantaj olmaktadır. Mesela robot bir tepe inişinde, hızlandığında ve ya 2m/s hızını aştığı zaman hata oranı affedilmeyecek ve göz ardı edilmeyecek şekilde yüksek olacaktır. Diğer yandan, vektör takip metodunda, yolun tamamında standart sapma oranı yaklaşık olarak sabit olmaktadır ve robot kararlı bir şekilde yolu izlemektedir.
İkinci senaryoda, robot hızı 1.5m/s olarak sabitlenip ve ön mesafe (0.5 , 1 , 1.3 , 1.7m) olarak değiştirilmektedir. Bu senaryoda da Saf Takip ve Vektör Takip sonuçlarına baktığımızda, Vektör Takip için standart sapma hatası yolun tamamında neredeyse sabit olmakta ama Saf Takip metodunda, ön mesafe arttıkça, hata ve aşım oranları da aşırı derecede artmaktadır.

Altıncı bölümde, tez içinde yapılan araştırmaların sonuçları ve bu konu üzerine gelecekte ne araştırmalar yapılabilir belirtilmiştir. Simulink sonuçları Vektör Takip metodunun Saf Takip metoduna göre daha güçlü ve daha verimli bir metot olduğunu göstermektedir. Ayrıca, Vektör Takip metodu, robot yolu daha kesin ve pürüzsüz şekilde takip etmektedir. Vektör Takip metodunun matematiksel karmaşıklıklarına rağmen, diğer metotlara tercih edilmektedir.

1. INTRODUCTION

This thesis describes methodologies in path tracking for a mobile robot that only works in indoor environments. The methods help robot to use the position information typically obtained from GPS or odometers and find an optimized path for the robot from the initial point to the goal point. Its goal is to autonomously drive the robot into the path, generate steering, and speed commands that correct the errors of tracking. In order, a robot tracks a specified trajectory; firstly, a path planning method must be introduced. Some path planning methods that has been used widely in the literature have been introduced. After specifying a certain path for the robot the next step is using path tracking algorithms.

1.1 Literature Review

There are several path planning methodologies on an environment indoor and outdoor map. For example, Dijkstra`s algorithm [1] is one of the efficient and basic short-path search methods that computes paths from every grid cell to the final destination and specifies a weight to each single sub-path on the path. Peter E. Hart et al. [2] proposed A* algorithm which is the advanced form of Dijkstra`s algorithm but it uses heuristics to improve its efficiency in finding the shortest path. Later of several modifications of A* have been discussed. A* algorithm does not support the path tracking in which the size of robot is larger than size of the cell, so Jitin Kumar Goyal et al. [3] proposed a new approach in which the virtual size of robot is assumed to be increased $(\boldsymbol{2n+1})$ times with respect to the size of cell. Modified A* method has been proposed by Chia-Jun Yu et al. [4] that considers the factors of distance and safety in the weight cost function, so the mobile robot reaches to its destination more quick and safe. Chabini and Lan [5] extends A* methodology to finding the shortest path in dynamic networks. Most of the works on the path planning assumes that the robot has a complete knowledge of the environment before it starts to move. Anthoney Stentz [6] proposed D*(Dynamic A*) method that is able to move in a partially known, unknown or changed fields. In this method, the robot finds its way to destination whenever it

discovers any obstacle on its way based on knowing the initial information and then modifying the plan locally or replan the whole trajectory as the robot finds obstacles with the help of its sensors. Nevertheless, this method sacrifices optimality and computational efficiency. Therefore, Sven Koenig et. al [7] proposed D* Lite method. This method behaves the same as D* method but is algorithmically different from D*. This method is simple, can precisely be analyzed, extendible and is as efficient as D*. In previous method, a point robot was considered. The methods mentioned does not cover the fact of being any obstacle on the path of robot mobile. Another method in Path tracking is visibility graph that is different in methodology from other methods in the shape of robot. Previously mentioned methods used a point robot in patch tracking, but this method is for Triangle robots [8]. This technique is just creating visibility graph of the environment that includes robot and obstacles, and connecting two vertices of obstacle that see each other and connecting the center of the robot (or a point robot) to the edges that is visible to it. Nora H. Sleumer et al. [9] presented exact cell decomposition method where the environment is composed of polygon shaped obstacles. This method results in connectivity graph that is famous in path planning in robotic science. The free space of the robot is exactly partitioned into cells that are stored in the graph. On the other hand, another method named Approximate Cell Decomposition is introduced in [10]. It also consists of robot`s free space $\boldsymbol{C_{free}}$ that is a collection of cells. It differs form exact cell decomposition in that the cells in this method must have a simple prespecified shape for example a rectangular shape.

Consequently, Path tracking methods uses path planning strategies for a mobile robot to track a specified path that has been planned or finding a short and efficient path, and follow it to reach from the initial point to the goal point. Martin Lundgren in [11] has used follow-the-carrot method for a 2-wheeled trajectory following robot. It is designed in a way that the robot maintains in its path every moment of its motion. In this method the robot exactly follows planned trajectory and does not deviate from it in any moment and in case it deviates, it tends to converge to its original trajectory. Another method that is widely devised in [12] [13] and [14] is pure pursuit method which operates like follow-the-carrot except for the fact that in pure pursuit method we fixate on a point some distance ahead on the path (look ahead distance) and try to follow it. It computes the arc necessary to return the rigid body from its current location back to the goal point somewhere on the path. Jesus Morales et al. in [15] has develop

pure pursuit method for reactive path tracking of autonomous nonholonomic vehicles that use 2D laser scanner on the robot to follow walls, corridors and persons autonomously. These two techniques only consider the position of a point ahead on the trajectory, and becomes nonsense when applied to these methods. Sir Robert Ball in 1900 developed a new approach of path tracking based on screw theory calls Vector Pursuit. This method generates a desired vehicle turning radius based on vehicle`s current position and orientation, relative to the position a certain distance ahead on the path and the desired orientation along the path at that point and has the ability to overcome large position and heading errors [16].

1.2 Motivation

Path tracking is an important fact in the autonomous navigations. Nowadays in many sciences that use robotics a path following is of a major importance. For example in medical world, very small robots are inserted into the veins to acknowledge the information in the body. Finding an optimal and efficient path results in saving time and money in long lasting attempts. The number of turning robots in the path also counts. Therefore, a path tracking controller should be inserted into a system and it must be robust to the tracking errors of robot. The geometry methods that are discussed in this thesis are already able to overcome that problem in different methodologies. Although, every one of them has its advantages and drawbacks that are going to be discussed in this thesis.

1.3 Methodology

In this thesis, In chapter 2 we have described some kinematics of the mobile robots. It is obvious that very robot has a virtual coordinate in name $\{\boldsymbol{X_R}, \boldsymbol{Y_R}\}$ with respect to the global frame. The position and orientation of the robot first must be determined using robots frame then it has to be converted in to the global frame so it can be able to keep in touch with other standard information. Each wheel on the robot has an important role in the behaviors of the robot. Therefore, the number of the wheels and positioning of them on the robot and their structure has a major role in the characteristic of a robot. In the following section in chapter 2, different kinds of wheels such as fixed wheels, steered standard wheels, Castor wheels, Swedish wheel and finally

spherical wheels has been described. Then different types of robot drives has been mentioned. Differential drive, Steered wheels, Synchronous drive, omnidirectional robots and Ackerman steering drives has been argued and their kinematics and dynamics has been studied.

In Chapter 3 a number of path planning methods in an unknown and known environments has been discussed for a mobile robot. At first Dijkstra`s shortest path finding method using weighted line segments is introduced. A* methodology which uses Dijkstra`s method is then descried. A dynamic form of A*, named D*, that uses heuristic path planning is described in the following section. These methods are useful in an already known terrain. For an unknown changeable terrain with obstacles in it another methods that are visibility graph and cell decomposition are introduced. The robot in these methods is not a point robot instead, it is a polygonal (for example triangle or square) one. In addition, obstacles has a polygon shape with known vertex coordinates and specified arcs.

After determining a specified trajectory for the robot to reach from its initial point to some goal point in chapter 4 we discuss about path tracking methods. A path tracking is also important in autonomous navigations because an accurate path tracking results in a safe and less time-consuming procedure. The most basic of all is follow-the-carrot, which aims to a goal point exactly on the pre-planned route or very near to it so the goal point is simply found by adding a particular value to the current x-coordinate and finding its related y-coordinate. Then the pure pursuit method that automatically tunes the look ahead distance, some distance ahead from the current position, based on path characteristics tracking errors and velocities is discussed and after that vector pursuit that uses the position and orientation of the goal point is mentioned. These methods have their drawbacks and advantages with comparison to the others.

2. KINEMATICS OF DIFFERET KIND OF ROBOTS

Studying robot behaviors needs to have a sufficient amount of information about kinematics of robots. Each wheel of a robot has an effect on how a robot acts and it imposes a constraint on the motion of robot. These constraints and forces must be expressed with respect to the global frame. A global frame and local frame of robot is defined in robotic sciences, and then we must have adequate equations to express the behavior of wheels and robots with respect to these frames. First, we began by expressing these frames, then introducing individual wheels and their dynamics finally the effect of the forces on the whole robot.

2.1 Robot Posture

We suppose a 2D horizontal plane with axis X, Y and a rigid mobile robot body on wheels. The Global frame is a 3-dimensional frame, the plane in which the robot moves has two dimensions and for the orientation of the robot, we have one dimension that is orthogonal to the plane in which the robot moves. To be able to determine the place of robot first we must be able to find a relation between global frame and local frame of robot (Figure 2.1).

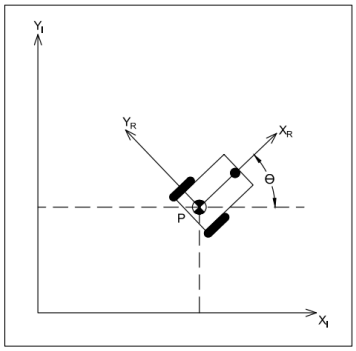


Figure 2.1 : The global frame and the local frame of robot.

The axes Y_I and X_I is a global reference with the origin $O:\{X_I, Y_I\}$. To specify the position of the robot we select a point P on the body of the robot. This point is the origin of the robot`s frame $P:\{X_R, Y_R\}$ with two axes X_I and Y_I. The pose of the robot

is described with three elements. Two of them are x and y that is the position of poit P and the third element is the angular difference between local and global frame named θ.

$$\xi = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{2.1}$$

By using the orthogonal rotation matrix it will be practical to describe and map robot motions in terms of component motions. This matrix helps to map motion along the axes of global frame $\{X_l, Y_l\}$ into the axes of robot`s local frame$\{X_R, Y_R\}$. The mapping will be accomplished using equation (2.2).

$$R(\theta) = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

This matrix is used to map motion in global frame $\{X_I, Y_I\}$ to motion in the local frame $\{X_R, Y_R\}$. It is obvious that this equation depends on a value of θ, therefore the operation is denoted by $(\theta)\dot{\xi}_I$. If we consider the position of the robot in figure 2.2, $\theta = \frac{\pi}{2}$, we can compute the relationship between the global frame and local frame of robot easily.

$$\dot{\xi}_R = R\left(\frac{\pi}{2}\right)\dot{\xi}_I \tag{2.3}$$

$$R\left(\frac{\tau}{2}\right) = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

If we consider the velocity of the robot in the global frame as $(\dot{x}, \dot{y}, \dot{\theta})$, we can compute the components of the motion along the robot`s local frame $\{X_R, Y_R\}$. In the case of figure 2.2 due to the angle of the robot, motion along X_R is equal to $\dot{y}$ and motion along Y_R is $-\dot{x}$. So we have the velocity of robot in equation 2.5 as:

$$\dot{\xi}_R = R\left(\frac{\pi}{2}\right)\dot{\xi}_I = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{y} \\ -\dot{x} \\ \dot{\theta} \end{bmatrix} \tag{2.5}$$

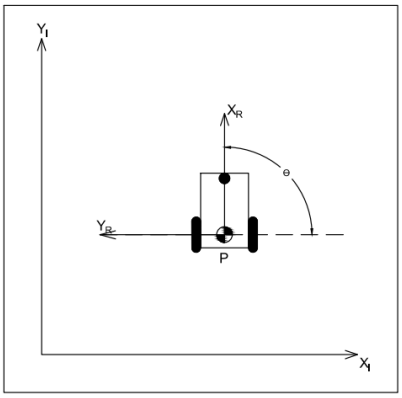


Figure 2.2 : The mobile robot aligned with a global axis.

2.2 Types of Wheels

The motion of each wheel can be combined to effect the motion of a robot as a whole therefore the first step to a kinematic model of the robot is to express constraints on the motions of each wheel. There are four basic wheel types that have variety of kinematic equations. Two assumptions must be considered first that the plane of wheel always remains vertical and second we assume that there is no any sliding phenomena and the wheel the wheel undergoes motion only in a case that there are pure rolling and rotation around the vertical axis. Considering these constraints there are two constraints for every type of wheel:

The first one is that the wheel must start to roll when motion takes place in the appropriate direction. Secondly the wheel must not slide orthogonally to the plane of wheel.

There are some fixed parameters for all types of wheels that has to be defined, r(radius of wheel), v (wheel linear velocity), ω (angular velocity of wheel).

2.2.1 Fixed wheels

In this kind of wheel as shown in figure 2.3.a the position of the wheel is defined in the local frame of robot and the distance from the wheel to the center of the wheel which is point P is defined as l, the angular difference between the orientation of the wheel A from X axis of local frame is shown by α, this means the polar position of the robot is shown by parameters l and α. The wheel can spin over time so its rotational posture around its horizontal axis is a function of time, $\varphi(t)$.

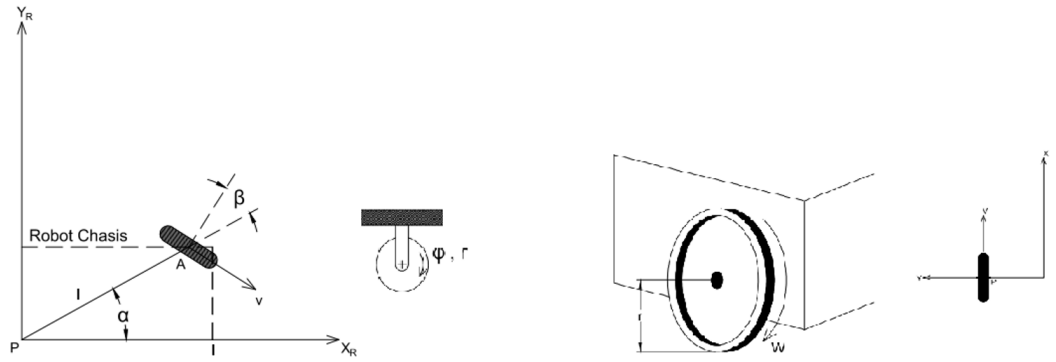
The angle of the wheel relative to the body of the robot is denoted by β. This value for the fixed wheels are fixed since the fixed wheel does not steer. The radius of wheel is

r, the angular velocity is ω, the linear velocity is defined by v as shown in figure 2.3.b and finally a_x is a unit vector to X_R axis. The fixed wheel has no vertical axis of rotation for steering and this is the restriction of this kind of wheel. Therefore, we can write the equation above for the fixed wheels:

$$v = (r \times \omega) a_x \tag{2.6}$$

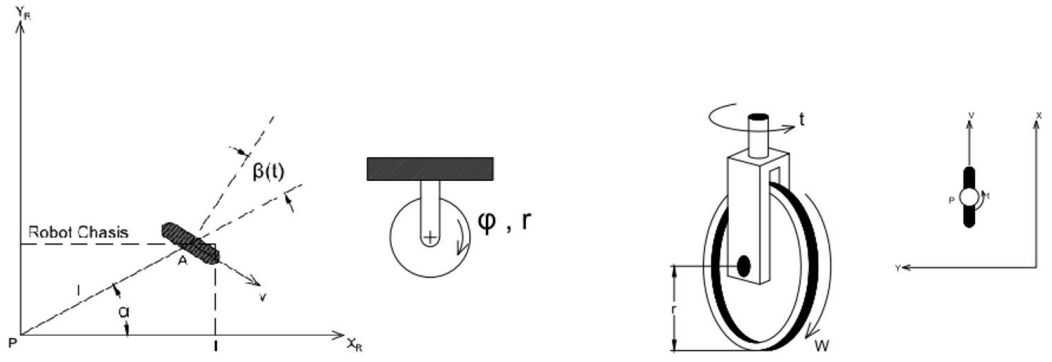
2.2.2 Steered standard wheels

Despite the fixed wheel, this wheel has two degrees of freedom. It means that the wheel can rotate also around a vertical axis that passes through the center of the wheel and the ground contact point. In this case, the value of the parameter β is not constant although it is a variant of time $\beta(t)$ as shown in figure 2.4.b. The parameters of this wheel are the same as the fixed wheel for $l, \omega, \beta, v, \varphi$ and α as shown in figure 2.4.a. The velocity of point P for the wheel is like equation (2.6).



a) b)

Figure 2.3 : a) Parameters of a standard fixed wheel b) 3D view of the wheel.



a) b)

Figure 2.4 : a) Parameters of a steered wheel b) 3D view of the wheel.

2.2.3 Off-Centered orientable wheels (Castor Wheels)

Castor wheels are able to steer around a vertical axis. However, unlike the steered standard wheel, the vertical axis of rotation in a castor wheel does not pass through the ground contact point. Figure 2.5.a depicts a castor wheel, demonstrating that formal specification of the castor wheel's position requires an additional parameter. The wheel contact point is now at position *B*, which is connected by a rigid rod *AB* of fixed length *d* to point *A*. *A* fixes the location of the vertical axis about which *B* steers, and this point *A* has a position specified in the robot's reference frame as in Fig. 2.5.a and a_y a unit vector of *Y*axis.

Figure 2.5.b shows the 3D aspect of this wheel in which by rotation around vertical axis passing through point *P* the orientation of the wheel changes so we have one extra equation added to the previous one that is dependent to the axis Y.

The castor geometry, however, has significant impact on the sliding constraint. The critical issue is that the lateral force on the wheel occurs at point *A* because this is the attachment point of the wheel to the body of the robot (chassis). Because of the offset ground contact point relative to *A*, the constraint that there be zero lateral movement would be wrong. Instead, the constraint is much like a rolling constraint, in that appropriate rotation of the vertical axis must take place so for the velocity at point *P* we have the equation (2.7):

$$V = (r \times \omega)a_x + (d \times t)a_y \tag{2.7}$$

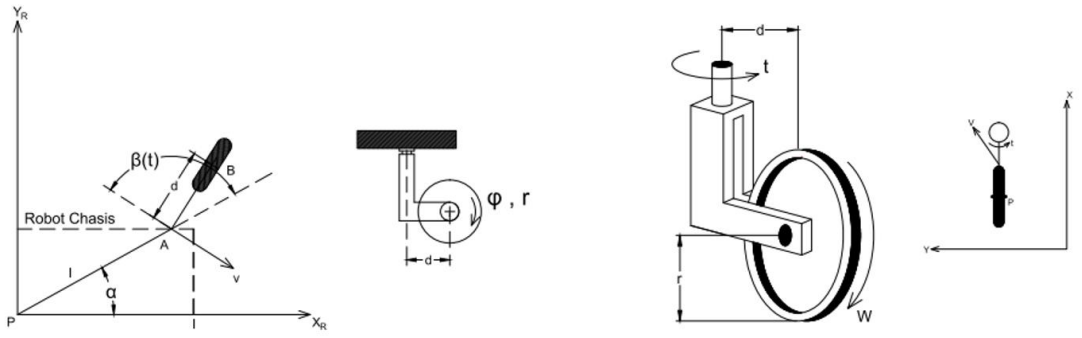
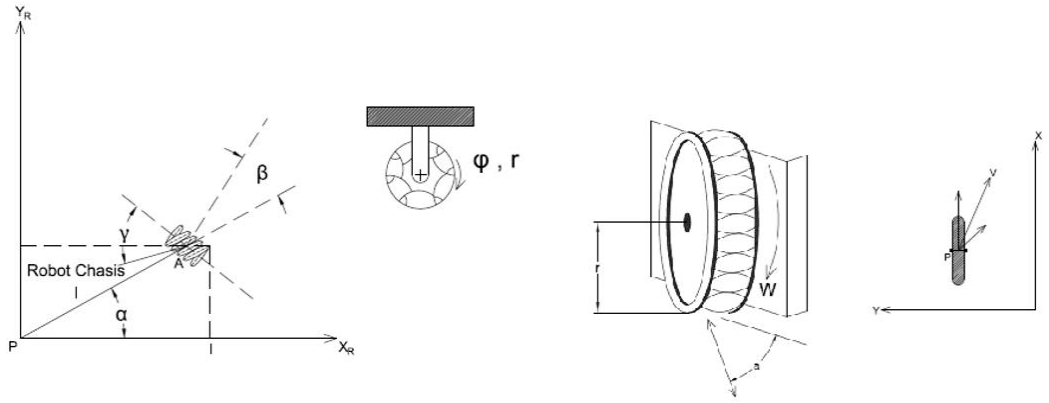


Figure 2.5 : a) Parameters of a castor wheel b)3D view of the wheel.

2.2.4 Swedish wheels

Swedish will is able of moving *omnidirectionally* like castor wheels but it does not have a vertical axis. This kind of wheel actually is a fixed wheel with one extra degree of freedom and that is the rollers that are attached to the wheel perimeter. Each one of these rollers have an axis that are not parallel with the main axis X. The angle between main wheel plane and the axis of the rotation of a roller is named by γ. This value,γ, can vary from one roller to another. As shown in figure 2.6.a. Since the axis of each roller can spin clockwise or counter-clockwise it is possible to combine these axes with each other and be able to lead the robot, notice that in order to be able to lead it in a desired direction on must select appropriate two vectors and combine them with each other (Figure 2.6.b). The angle between main wheel plane and the axis of the rotation of a roller is named by γ. The velocity in point P,therefore, achieves from equation (2.5). The second term of the equation results from the one degree of freedom added to the wheel and that is the orientation of the roller, there for the parameter a_s in the equation defines a unit vector to the motion of the roller.



a) b)

Figure 2.6 : a) Parameters of a castor wheel b) 3D view of the wheel.

2.2.5 Spherical wheels

The last type of wheel is a ball or spherical wheel that has no constraints on the motions of the robot. This kind of wheel has not specific axis of rotation and therefore it has no specific rolling or sliding constraints. Like castor wheels and Swedish wheels this kind of wheel is an omnidirectional one on robots motion kinematics (Figure 2.7).

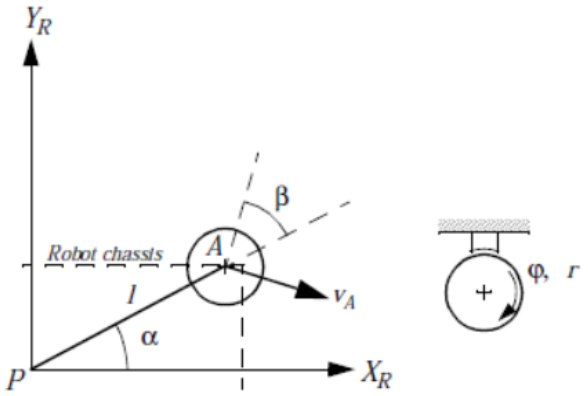


Figure 2.7 : spherical wheel and its parameters.

2.3 Mobile Robot Maneuverability

The maneuverability of a robot depends on some characteristics of a robot and its wheels. Those factor are degree of mobility and degree of steerability.

2.3.1 Degree of mobility

Consider a single standard wheel. It is forced by the sliding constraint to have zero lateral motion. This can be shown geometrically by drawing a zero motion line through its horizontal axis, perpendicular to the wheel plane (Figure 2.8). At any given instant, wheel motion along the zero motion line must be zero. In other words, the wheel must be moving instantaneously along some circle of radius R such that the center of that circle is located on the zero motion line. This center point, called the Instantaneous Center of Rotation (ICR), may lie anywhere along the zero motion line. When R is at infinity, the wheel moves in a straight line.

The car in figure 2.8.a can have several wheels but must always have a single ICR. The geometrics of ICR shows how a robot is dependent to the number of constraints on robot`s motion, not the number of wheels.

Figure 2.8.b shows a bicycle that has two wheels, these wheels are totally independent from each other so they have separate zero motion line and the conjunction of these two lines results in just one single ICR for the robot`s constraint.

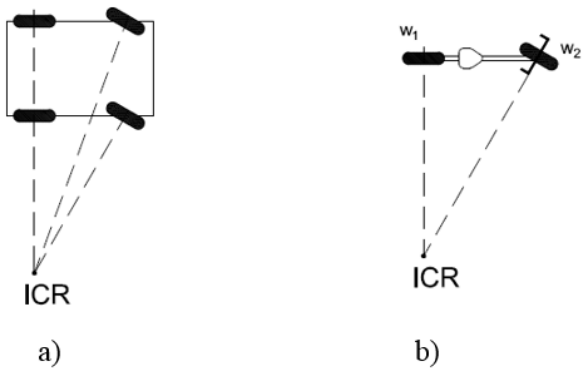


Figure 2.8 : a) Ackermann steering b) bicycle.

To understand the problem more accurately consider figure 2.9. There are several kinds of wheels on the body of the robot. In (a) there are three independent wheels that are attached to the chassis of the robot. The robot have to start moving on an arc created from the ICR of the zero motion lines of these three wheels. The ICR point is actually the center of this circle. Drawing zeromotion line of these there wheels, we can see that there is no common point for them to create an ICR. Therefore, this robot cannot move anywhere and has zero degree of mobility. In (b) because both back wheels and forward wheels will move parallel to each other, therefore they create zeromotion lines that meet at a point named ICR. So, this robot is able to move on an arc that is the part of circle originated with ICR. The degree of freedom of this robot is one and has only one ICR. In (c) There are infinite number of points that the zeromotion line of these two wheels meet. So it has variable arc motions (ICRs) and the degree of the mobility is two. In (d) because each wheel is dependent from another there will be three degrees of freedom and the ICR can be located at any position, this means that the robot is able to move in any desired motion.

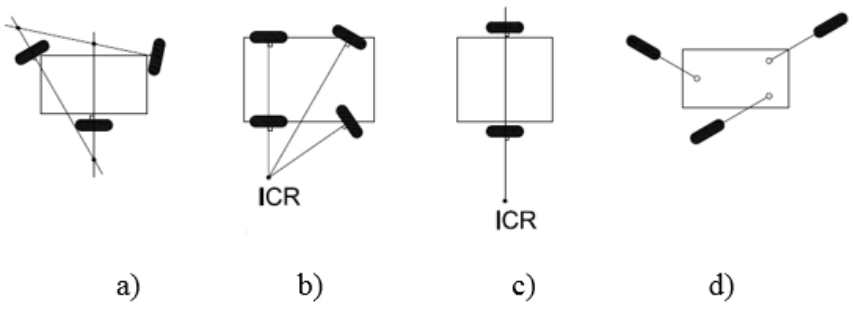


Figure 2.9 : Different kinds of ICRs.

2.3.2 Degree of steerability

Number of steerable standard wheels that can be steered independently is degree of steerability(DOS). Steering can have an impact on a robot chassis pose. Looking at figure 2.10.a it is seen that there are no centered oriented wheels therefore degree of steerability for both bodies are zero. In figure 2.10.b there is only one centered orientable wheel so DOS is one. For figure 2.10.c here are two mutually dependent centered orientable wheels so DOS will be two and for the last one in figure 2.10.d there are two mutually independent centered orientable wheels therefore DOS will be two.

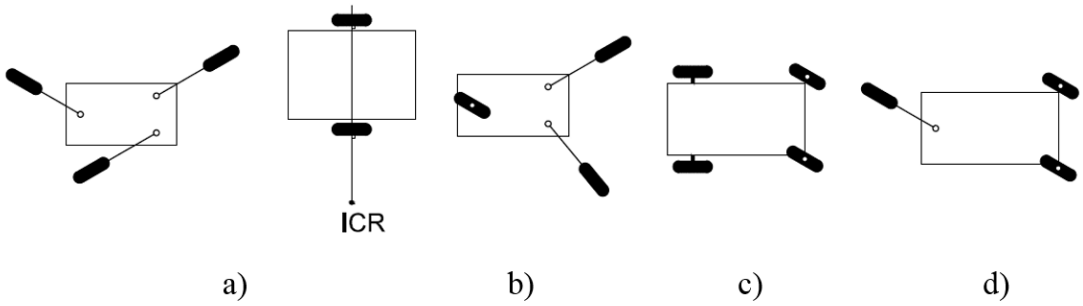


Figure 2.10 : Various types of robots to show degree of steerability.

2.3.3 Degree of Maneuverability

The overall degrees of freedom that a robot can manipulate is called degree of maneuverability δ_M and can be defined in terms of mobility and steerability (equation 2.8). Therefore maneuverability includes both the degrees of freedom that the robot manipulates directly through wheel velocity and the degrees of freedom that it indirectly manipulates by changing the steering configuration and moving.

$$\delta_M = \delta_m + \delta_s \tag{2.8}$$

So, maneuverability is the combination of the degrees of freedom that the robot manipulates directly through wheel velocity and degrees of freedom that indirectly manipulates by changing the steering configuration and moving. Based on what we have investigated before basic types of wheel configurations are depicted in figure 2.11. Mentioning that the same number of degree of maneuverability does not mean that two robots are the same. For example for differential and tricycle modes degree

of maneuverability is two $\delta_M = 2$. In Tricycle maneuverability is the combination of direct motioning $\delta_m = 1$ and steering$\delta_s = 1$. Nevertheless, in differential mode maneuverability is all result of mobility because $\delta_m = 2$. In the case of differential drive, this line extends from the common axle of the two fixed standard wheels, with the differential wheel velocities setting the *ICR* point on this line. In a tricycle, this line extends from the shared common axle of the fixed wheels, with the steerable wheel setting the *ICR* point along this line. Shortly, for robots with $\delta_M = 2$ the ICR is always lying on a line and for $\delta_M = 2$ the ICR can be set to be on any point on the plane of motion.

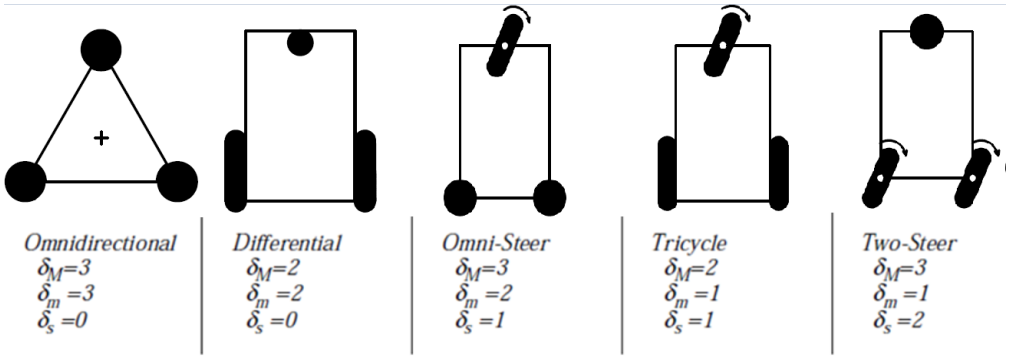


Figure 2.11 : Five types of three-wheel configuration

2.4 Mobile Robot Locomotion

There are several types of mobile robots that change in number of wheels, types of wheels and the situation of wheels. These differences results in completely different characters for each mobile type. Some types of robots that are going to be introduced and discussed its kinematics is listed below:

1. Differential Drive
2. Synchronous Drive
3. Omni-directional
4. Car Drive(Ackerman Steering)

2.4.1 Differential drive

A differential wheeled robot is a simplest driven mechanism in a robot which its movement depends on two separately driven wheels that can independently being driven either forward or backward. These two wheels are located on either side of body of the robot and mount on a common axis. The direction of the robot can be changed by variation of the relative rate of robot wheels.

If both of the wheels start to move in the same direction and speed then the robot will move on a straight line. By changing the velocity of each wheel the robot body will start to move one a curvature with the center point that lies along the common axis of left and right wheels. This point is known as *Instantaneous Center of Curvature* (ICC) as shown in Figure 2.12.

The velocity of rotation ω of two wheels around ICC must be the same in order the robot moves on a straight line. By varying thee velocity of two wheels the trajectory of the robot will change. These two velocities for the right wheel and left wheel are shown below as V_r and V_l respectively where l is the distance between two center of wheels and R is the distance from /ICC to the midpoint of the wheels.

$$\omega \left(R + \frac{l}{2}\right) = V_r \tag{2.9}$$

$$\omega \left(R - \frac{l}{2}\right) = V_l \tag{2.10}$$

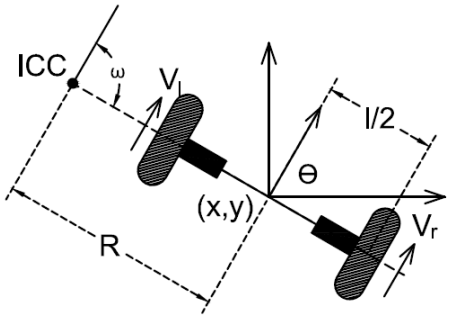


Figure 2.12 : Differential Drive Kinematics of a mobile robot.

Depending on the speed of rotation and its directions the place of ICC will vary along the line defined by two contact points of the tires. Since the direction of the robot is dependent on the speed of the two wheels, so a slightest error between the speeds of two wheels will cause error in the trajectory. Therefore, these quantities must be sensed

and controlled precisely. If we consider the speed of the left wheel (r) less that right wheel($r + b$) the robot will start to steer left (Figure 2.13).

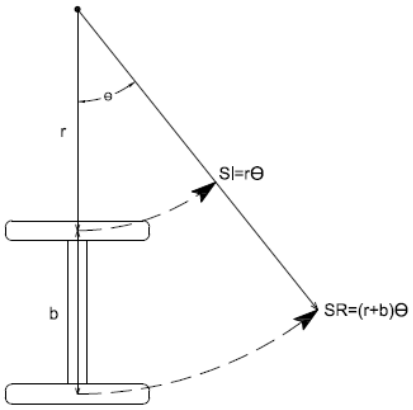


Figure 2.13 : Path of wheels through left turn

At any distance we can find velocity and we can find R and ω as:

$$R = \frac{l}{2}\frac{V_l + V_r}{V_r - V_l} \tag{2.11}$$

$$\omega = \frac{V_r - V_l}{l} \tag{2.12}$$

There are three interesting cases for these kind of drives;

1. If $V_r = V_l$, then we have a linear forward motion and an infinite quantity for R and the place of ICC will be at infinity and because there is no rotation $\omega = 0$.
2. If $V_r = -V_l$, the robot will rotate in place and there is no R.
3. If $V_r = 0$, then we have rotation around right wheel, The same is also true for $V_l = 0$. In these two cases $R = \frac{l}{2}$

In figure 2.12, assume the robot is at some positon (x, y), headed in a direction making an angle θ with the X axis. We assume the robot is centered at a point midway along the wheel axle. By manipulating the control parameters Vl , Vr, we can get the robot to move to different positions and orientations. (note: Vl , Vr) are wheel velocities along the ground).

Using equations 2.11 and 2.12 we can find the ICC location:

$$ICC = [x - Rsin(\theta), y + Rcos(\theta)] \tag{2.13}$$

And the kinematic equation of the differential driven robot will be :

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix}=\begin{bmatrix}cos\theta & 0\\ sin\theta & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}v\\ \omega\end{bmatrix} \tag{2.14}$$

Where v and ω are find as:

$$\omega=\frac{V_r-V_l}{l}$$

$$v=\frac{V_r+V_l}{l}$$

$$V_l=r\,\omega_l \qquad\qquad V_r=r\,\omega_r \tag{2.15}$$

Also kinematics model in robot frame in the function of angular velocity of right and left wheel.

$$\begin{bmatrix}V_x(t)\\ V_y(t)\\ \theta(\dot{t})\end{bmatrix}=\begin{bmatrix}r/2 & r/2\\ 0 & 0\\ -r/l & r/l\end{bmatrix}\begin{bmatrix}\omega_l(t)\\ \omega_r(t)\end{bmatrix} \tag{2.16}$$

And at time $t+\delta t$ the robot`s pose will be:

$$\begin{bmatrix}V_x(t)\\ V_y(t)\\ \theta(\dot{t})\end{bmatrix}=\begin{bmatrix}\cos(\omega\,\delta t) & -\sin(\omega\,\delta t) & 0\\ \sin(\omega\,\delta t) & \cos(\omega\,\delta t) & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x-ICC_x\\ y-ICC_y\\ \theta\end{bmatrix}+\begin{bmatrix}ICC_x\\ ICC_y\\ \omega\,\delta t\end{bmatrix}$$

The above equation describes the motion of a robot rotation a distance R about its ICC with an angular angular velocity ω.

2.4.2 Synchronous drive

As shown in figure 2.14 in this type of drive each wheel is capable of both steering and driving and is consisted of three wheels that has been arranged at the vertices of an equilateral triangle and is surrounded by a cylindrical platform. In synchronous drive all the wheels always point the same direction and turn at the same rate and all of the wheels turn and drive unison. A steered wheel is a wheel for which the orientation of the rotational axis of the wheel can be controlled. This unison moves typically is accomplished from the use of complex collection of belts that physically link the wheel together. In this drive robot, the vehicle controls the rate in which the robot roll and direction in which the wheels point.

In this type there is two separate motors, one of which controls the rotation of the wheels and the other rolls the wheel forward.

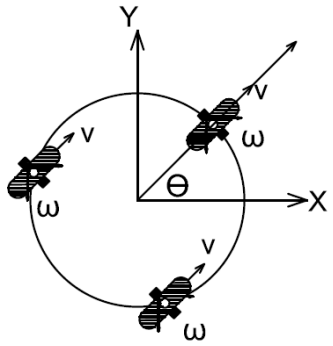


Figure 2.14 : Synchronous drive with three wheels.

Since all the wheels remain parallel, this drive always rotates around the center of robot. Thus, it has the ability to directly control the direction of θ. Synchronous drive robots rotate about the center at the rate of ω and the translational speed v. The forward kinematics of this type of drive is obtained in equation (2.17.)

$$x(t) = \int_0^t v(t)\cos\big(\theta(t)\big)\,dt$$
$$y(t) = \int_0^t v(t)\sin\big(\theta(t)\big)\,dt$$
$$\theta(t) = \int_0^t \omega(t)dt \qquad (2.17)$$

In this case the ICC for robot is always at infinity and changing the orientation of the wheels manipulates the direction of ICC. In some particular cases if $v(t) = 0$ and $\omega(t) = \omega$ during a time interval Δt, then the robot rotates at its place with the rate of $\omega\,\Delta t$. In other case if $v(t) = v$ and $\omega(t) = 0$ then the robot moves forward to the point in a direction at the distance $v\,\Delta t$.

2.4.3 Tricycle

A typical tricycle drive has three wheels that has odometers on two rear wheels. The steering and power of motion is provided by the front wheel. As shown in figure 2.15 the velocity v and steering direction α is provided through the front wheel. If the steered front wheel is set at the angle α from the straight direction, the tricycle will rotate with angular velocity ω about a point lying a distance R along the line perpendicular to and passing through the rear wheels, then we have:

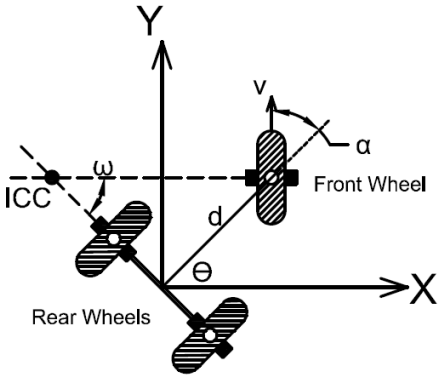


Figure 2.15 : Tricycle schematic

$$v_s(t) = \omega_s(t)r$$

$$R = d\tan(\frac{\tau}{2} - \alpha)$$

$$\omega = v/(d^2 + R^2) \;\Rightarrow\; \omega = \left(\frac{v}{d}\right)\sin\alpha(t) \tag{2.18}$$

And d is a distance from the rear to the rear axle as shown in figure 2.15. The kinematics of the tricycle can be shown in the robot frame and in the world frame. Here we have both of these kinematics in equations 2.19 and 2.20 respectively.

$$v_x(t) = v_s(t)cos\alpha(t)$$

$$v_y(t) = o \quad (noslippage)$$

$$\dot{\theta}(t) = \frac{v_s(t)}{d}\, sin\alpha(t)$$

$$\dot{x}(t) = v_s(t)cos\alpha(t)cos\theta(t)$$

$$\dot{y}(t) = v_s(t)cos\alpha(t)sin\theta(t)$$

$$\dot{\theta}(t) = \frac{v_s(t)}{d}\, sin\alpha(t)$$

$$\begin{bmatrix}\dot{x}(t)\\ \dot{y}(t)\\ \dot{\theta}(t)\end{bmatrix} = \begin{bmatrix}cos\theta(t) & 0\\ sin\theta(t) & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}v(t)\\ \omega(t)\end{bmatrix} \tag{2.19}$$

$$\omega(t) = \frac{v_s(t)}{d}sin\alpha(t) \quad , \quad v(t) = v_s(t)cos\alpha(t) \tag{2.20}$$

2.4.4 Omni-directional robots

Consider the omni-robot shown in figure 2.16. These kind of robots has three Swedish type wheels that are arranged radially symmetrically and their roller is perpendicular

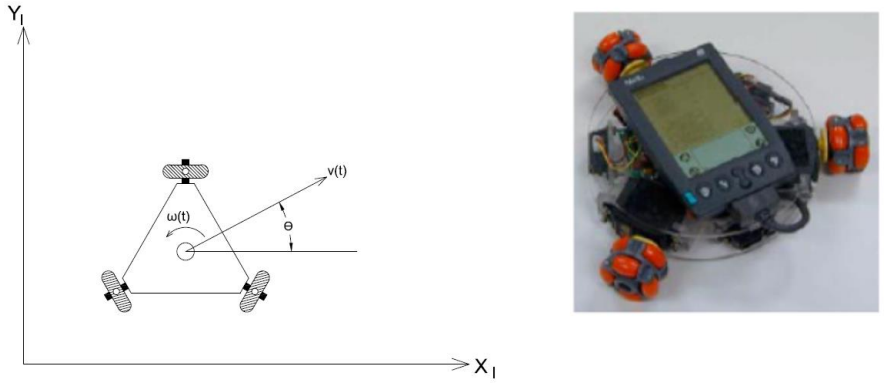


Figure 2.16 : A three-wheel omni-drive robot developed by Carnegie Mellon University.

to each main wheel. The first step is defining any robot is imposing a specific local reference frame for the robot. Let`s consider the point P at the center of a robot. Assume the distance between the center of each wheel and point P as l and all wheels has the same radius of r (Figure 2.16).

All three wheels are able to move rotationally and in a straight line, therefore this robot is able to move instantaneously in any direction. It can move in a direct straight line,

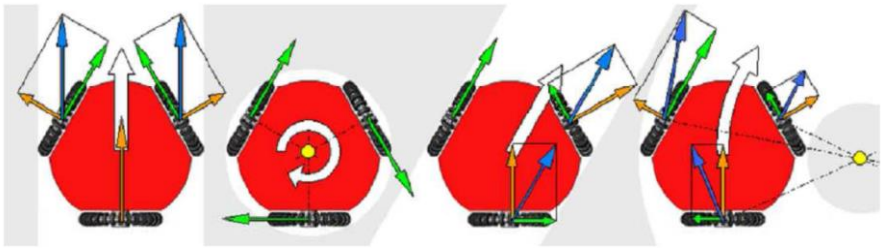


Figure 2.17 : Different directions of a moving omni-directional wheel

can move in its location around itself, it can move on an inclined straight line and it is able of moving on a curvature as shown in figure 2.17 respectively from left to right.

2.4.5 Car drive (Ackerman steering)

An Ackerman drive is used in motor vehicles as cars and it consists of four wheels, two of them in front and two in the back of the vehicle. In this kind of vehicle as shown in figure 2.18, when the vehicle starts to turn to the right or left the associated front wheel will rotate slightly sharper than the outside wheel and the tire slippage will be less in result. The center of rotation for both front wheels that is ICR, is a same point on the side of the inside wheel. The main intention of Ackerman geometry is to avoid wheels from sliding sideway when following the path around a curve. Though, This method is used generally in outdoor autonomous vehicles.

Two backward wheels are joint to each other with a gear at a longitudinal distance d. The front axis and back rear, on the other hand, have a lateral distance of l. The reference point that all the values are measured from that point, is a point along the line that connects two back wheels on the rear of the vehicle to each other. The relative steering angles of inside wheel and outside wheel, with respect to the line perpendicular to the forward-axis of the vehicle, is defined by θ_i and θ_o respectively. The radius of rotation of the car on a circle's perimeter with the center of ICR, is R (figure 2.18).

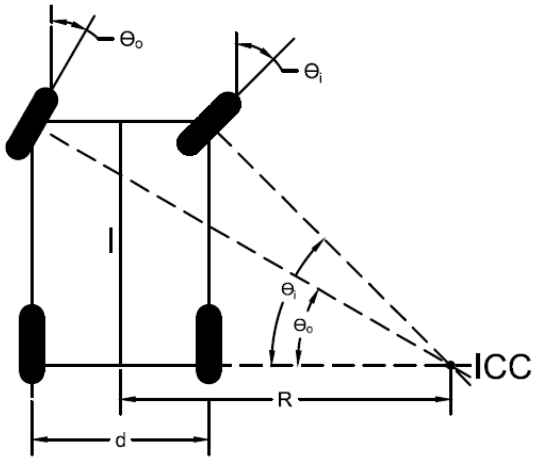


Figure 2.18 : An example of Ackerman drive.

Therefore, the relationship between these parameters on an Ackerman steering drive can be described as in equation 2.21.

$$\cot\theta_i - cot\theta_o = \frac{d}{l} = \frac{R-\frac{d}{2}}{l} - \frac{R+\frac{d}{2}}{l} \tag{2.21}$$

and the steering angle φ is given by

$$\tan\varphi = \frac{l}{R} \tag{2.22}$$

The refrene point of the Ackerman vehicle follows a circular trajectory and its angular velocity is defined by $\dot{\theta}$ and the turning radius R_1 is as:

$$\dot{\theta} = \frac{v}{R_1}$$
$$R_1 = \frac{l}{\tan\varphi} \tag{2.22}$$

Let`s consider robot coordinate system $(x_r, y_r, \theta_r)$ is centered in the middle of rear axis. Then $\dot{x}_r$ in the linear velocity of the robot that is the produced by vehicle`s engine. Now this is possible to derive expresions for the vehicle`s position in world coordinates (x_w, y_w, θ_w). The velocity of robot in the world frame is

$$\begin{aligned} \dot{x}_w &= cos\theta_w \, \dot{x}_r \\ \dot{y}_w &= sin\theta_w \, \dot{x}_r \\ \dot{\theta}_w &= \frac{tan\varphi}{l} \dot{x}_r \end{aligned} \tag{2.23}$$

This is important to calculate the resulting wheel angles and have a knowledge of kinematics of an Ackerman drive.

This model is referred as kinematic model of the Ackerman drive since it describes velocities of the vehicle but not the torques or forces that cause this velocity. The rate of change of angular velocity $\dot{\theta}$ is referred as turn rate, heading rate of raw rate and this value can be measured by gyroscope. This can also be resulted from angular velocity of each wheel on the right side or left side of the vehicle. Hence they have arcs of different radius and therefore rotate at different speeds.

The vehicle cannot move along y axis in the world coordinate so it has a non-holonomic constraint and we can write and expression for velocity in the vehicle`s y-direction as

$$\dot{y}cos\theta - \dot{x}sin\theta = 0 \tag{2.23}$$

From equation 2.23, when $\dot{x}_r$=0 then angular velocity with respect to world coordinates is $\dot{\theta}_w = 0$, that is , it is not possible for the vehicle to change its orientation while it is not moving, actually as we know, a drive should be moving in order to turn.

3. PATH PLANNING MTHODS FOR MOBILE ROBOTS

Path planning is a term used in the field of mobile robots in order to let the autonomous mobile robots to find their way in a reliable trajectory by breaking down the desired path into some discrete cells of motion that helps the robots with its constraints and possibly optimize the moving aspects. These constraints can be the speed, rotation orientation and turning commands that are sent to the mobile robot.

Briefly, path planning is a problem in which mobile robots find out how to navigate from point A to point B without colliding with obstacles and walls in the shortest way and shortest amount of time. There are various types of path planning methods for mobile robots. For example, A* (read A star) algorithm [2-4], Dijkstra`s algorithm [3], D* method, visibility graph method and cell decomposition. These methods are briefly described in the following sections.

3.1 A* Algorithm

A* method is a traditional and widely known method in path planning of autonomous mobile robots that focuses on finding the shortest path. A* search begins by expanding the node which it starts in and placing all of its neighbor cells in a stack. These cells start to find the shortest path between starting node s_0 and final node G. The distance between the place of robot on the surface from starting node $s_0 = (x_0, y_0)$ is called the movement cost $g(n)$. The heuristic estimation of the minimum cost of a path from the current expanding cell n to the goal node $G = (x_G, y_G)$ is that is the Euclidean distance is denoted by $h(n)$. This distance can be taken as Manhattan distance or any other distance for the path planner. Therefore, the cost evaluation function, which helps the robot find the shortest path, is given by

$$f(n) = g(n) + h(n) \tag{3.2}$$

Where $n = (x_n, y_n)$ is the current expanding node in the surface and $f(n)$ is the estimated function of the minimum path along all pathes that start in node S and ends up in node G.

$g(n)$ is defined as

$$g(n) = \begin{Bmatrix} dist(prev(n), n), & if\ prev(n) = s_0 \\ g(prev(n)) + dist(prev(n), n) else & \end{Bmatrix} \quad (3.2)$$

Where prev (n) (read as previous n) is the current node of k step and is denoted by s_k, so

$$\text{Dist(prev(n),n)} = \text{dist}(s_k, n) = \sqrt{(x_k - x_n)^2 + (y_k - y_n)^2} \quad (3.3)$$

The surface, which the robot moves in, is actually gridded in cells as shown in figure (Figure 3.1). Direct distance of two nodes is denotes by d and diognl distance of two nodes is denoted by $\sqrt{2}d$.

$h(n)$ is defined as

$$h(n) = d \times \sqrt{(x_n - x_G)^2 + (y_n - y_G)^2} \quad (3.4)$$

$\sqrt{2} \cdot d$	d	$\sqrt{2} \cdot d$
d	node	d
$\sqrt{2} \cdot d$	d	$\sqrt{2} \cdot d$

Figure 3.1 : Distance relation between two nodes.

Now the lowest cost solution of the path $p_0 - p_1 - p_2 - \cdots - p_N$ can be find from the above equations and back tracked from the goal point G to the start node S where $p_0 = G$ and $p_N = S, N \leq K$ [17].

3.2 Dijkstra`s Algorithm

In Dijkstra`s algorithm is a single-source shortest path algorithm in which we demonstrate two sets, the first set consists of vertices that are included in shortest path tree, the second set includes vertices that are not yet included in this tree. At each step we find new vertices that are not already in the other set (set of not yet included) and has a minimum distance from the destination node.

3.2.1 Directed-weighted graph

Directed graph can be expressed in the form of $G = (V, E)$. In this ordered pair the elements of V are called as the nodes or vertices and the elements of $\boldsymbol{E}$ are called as edge cost of the ordered pairs of vertices that is shown in the form of arrows or arcs (Figure 3.2). The cost number written on each arrow can be defined to be energy expended, distance travelled and time exposed to danger.

Directed weighted graph is an advanced directed graph in which the edge of the graph has its weights on it shown with numbers on the arrows (Figure 3.3).

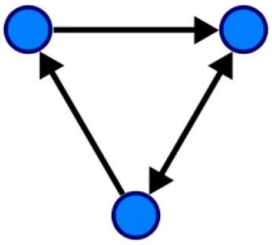


Figure 3.2 : Directed graph.

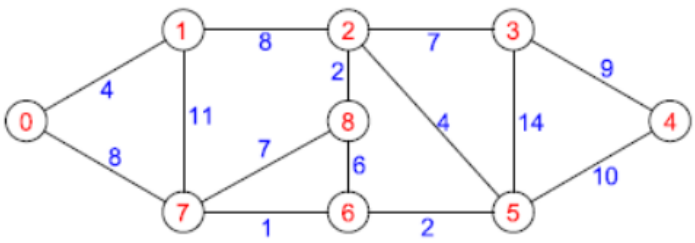


Figure 3.3 : Directed weighted graph.

3.2.2 Algorithm`s definition

Below there are detailed steps of how this algorithm iterates in each step.

1) To keep track of the vertices that include in the shortest path tree, we create a set named *sptSet* (shortest path tree set). The algorithm works by keeping the shortest distance of vertex *V* from the source in an array.

2) Firstly, this set is empty and the distance value for the source vertex is set as zero and for the other vertices that are not yet processed this value is set to infinity, but then it starts to gain elements whose minimum distance from source node is calculated.

3) While *sptSet* does not include any elements pick a vertex, which has a minimum distance value. Include this number in the *sptSet* and then update all distance values for the neighbor vertices. For every neighbor vertex u,if the sum of weight of edge *u-v* and distance value *u*, is less than the distance valued of *v*, then update the distance value of *v*.

3.2.3 An example of Dijkstra`s algorim

Considering figure (3.4), the sptSet is initially empty it means that the elements of the set is {0, inf, inf, inf, inf, inf, inf, inf} where “inf” indicates infinity. The first element of the set is the value of the source that is zero. Therefore, the value of first vertex is entered into the set and sptSet becomes {0}. Now update distance values of the adjacent vertices that are 1 and 7 and their distance value are 4 and 8 respectively (Figure 3.4).

Subgraphs shown below, shows vertices and their distance value and the vertices included to SPT are shown in green color.

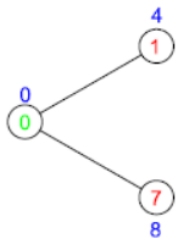


Figure 3.4 : First step in Dijkstra`s weighted graph.

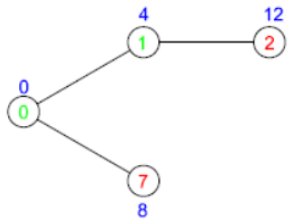


Figure 3.5 : Second step in Dijkstra`s weighted graph.

We have to pick a minimum distance value that is not already in SPT. The vertex 1 has a value of 4, then it is selected and added to SPT so the new form of the set is {0,1} so the circle goes green (Figure 3.5). Now update the distance values of adjacent vertices of 1. Therefore, the distance value of vertex 2 becomes 12. From the vertexes that has not gone green, vertex 7 and 12(not included in the SPT) choose the one, which has the less distance value, and update the *sptSet*.So we choose the 7 vertex and now *sptSet* becomes {0, 1, 7}. The adjacent vertices to 7 are 6 and 8 and their distance value will be 15 and 9 respectively (Figure 3.6).

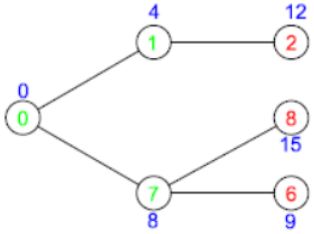


Figure 3.6 : Third step in Dijkstra`s weighted graph.

Again, choose the vertex that has the minimum distance value and is not included in sptSet. Vertex 6 has the value of nine that is less that the other two, so pick vertex 6. The new sptSet is now {0, 1, 7, 6}. Update the distance value of vertex 5 and 8. Therefore, their distance value will be 11 and 15 respectively (Figure 3.7).

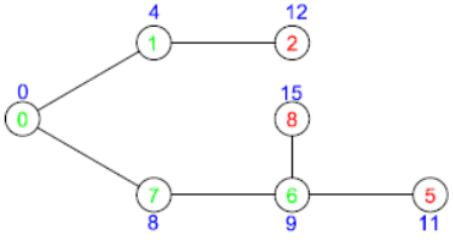


Figure 3.7 : Fourth step in Dijkstra`s weighted graph.

This procedure iterates for all vertices by finding the least value of selected vertices until sptSet does not include all vertices of the given graph. At last, we have the following shortest path tree (Figure 3.8).

The last set consists of 5 members from the start node to the end node which are {0, 1, 7, 6, 5, 4} and has the least distance value in Dijkstra`s algorithm.

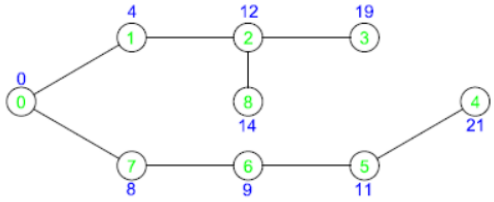


Figure 3.8 : Last step in Dijkstra`s weighted graph.

3.3 D* Lite Algorithm

The above methods are used for the already known environments and less attention has been paid to the problem of partially known environments, however, in real life scenarios, robots do not have all the information about paths. They may have an incomplete information or inaccurate planning graphs. Actually any path planning that has been directed at the first place for the mobile robot may be invalid as robots moves along the path and gets updated information of the trajectory. A sensor on the robot has the ability of detecting obstacles on the path for instance and it is important that robot is able to update its information through its way by the means of this sensor attached to the body of the robot.

Having the updated graph, a new optimal path can be make using above A* method. An approach for this replanning is to replan the path that has been crossed over and named as scratch. However, this method can be very expensive because it recomputes the path every time that an obstacle is detected. In some points, this replanning is useless. For example, imagine a change that occurs in the graph but this change will not affect the optimality of the current graph. Alternatively, another example, imagine the changes that has been occurred, does not affect the graph of the path, but in a very simple way that can be quickly fixed. As described in examples, it is a waste of time to compute and replan from scratch. Instead, it is more useful and efficient to take the previous solution and make it right to account for the changes to the path graph.

Existing approaches mentioned in previous sections based on known information scarify optimality and computational efficiency. This new algorithm D*, however, introduces a new way of path planning in which for the unknown of partially known paths the robot reaches an efficient and optimal path.

Many algorithms have been discovered for this repair. The original D*, D* Lite [18] and focused dynamic A*(D*) [19] are the widely used algorithm these days, because they are more efficient in incremental and heuristic updates. These two methods are the extensions of A* method and have been developed for planning a wide range of robots` path in outdoor and indoor environments. They have also been extended to raplan incrementally in symbolic planning domains.

These three algorithms solve the same problem in path planning, where the robot has to reach to the goal point in an unknown terrain. It makes assumptions about unknown parts of the trajectory, say walls, obstacles, etc. and finds the optimal short path to the goal coordinates. It detects obstacles and if necessary replan the route until it reaches to the destination. While new obstacles may be found rapidly on the path, the prices of finding them must be fast too. Therefore, heuristic (incremental) search algorithm speeds up the procedure. Considering fix coordinate of the goal position, these three algorithms are more efficient that repeated A* searches in the path.

D* and its sub-methods have been widely used for autonomous vehicle navigation and mobile robots. Current methods are generally based on D* Lite method rather that original D* or Focused D*.

The original D* comes from "Dynamic A*" and was introduced by Anthony Stenz in 1994 [20]. This algorithm is like Dijkstra`s and A*, it maintains a list of nodes to be evaluated, namely "OPEN list'. They are several states for the nodes namely: NEW, means it has not yet been included to OPEN list; OPEN, means it is currently on the OPEN list; CLOSED, means it is no longer on the OPEN list; RAISE, its cost is now higher that the last time it was on the OPEN list; LOWER, its cost is lower that the last time it was in OPEN list.

D* algorithm is different from A* in the way that D* begins by searching from the goal point in the opposite manner that A* follows the path from the start to the destination. Each node has a back pointer, which refers to the next node that leads to

the destination, and each node knows the cost to the target. The algorithms is done, when the start node is the next node to be expanded and the path to the destination can be found by following the back pointers.

The members of the OPEN list varies when an obstacle occurs on the path, it means all the points that are effected by the obstacle are placed on the OPEN list again and are marked as RAISED. While a RAISED node can be reduced, the back pointer is updated and marks its neighbors as LOWER state. The D* is made by this RAISE and LOWER states and by this waves a whole series of other points are prevented to be "touched". We can see that the algorithm just worked on the points that is affected by change of costs.

3.4 Visibility Graph

Visibility graph method which introduced in Shakey project at SRI in the late 60s is a method which is applied in 2D world with polygon shaped obstacles and its aim is to find the shortest trajectory between the start point and destination by moving the polygon shaped robot. This method is different from previous methods in the shape of robot. Another problem to be concerned here is the shape of obstacles in this method. All the obstacles in this method are polygons [21].

3.4.1 Trapezoidal map

Working on the 2D environment for the visibility graph requires defining some definitions and separate the work space and configuration space into some regions so the robot can be able to find the shortest path using these regions.

Work space and configuration space

Let $\mathcal{R}$ be a robot moving in a 2D environement or *works pace* that is a simple polygon with obstacles set of $S = \{\mathcal{P}_1, \dots, \mathcal{P}_N\}$. A *configuration* or the place of the robot can be specified by translation vector. We denote the translation vector symbol of the robot positioning at the point (x, y) by $\mathcal{R}(x, y)$. For example, if the robot is a polygon with vertices (-1,-1), (-1, 1), (0, 3), (1, 1) and (1,-1) and we want to move the robot due to the translation vector $\mathcal{R}(6,4)$ then the new coordinates of the vertices of the robot will be (5,3), (5,5), (6,7), (7,5) and (7,3) as in figure 3.9. The reference point of the

translation is denoted by $\mathcal{R}(0,0)$ and in this case the reference point is somewhere in the robot but in general the reference point does not have to be in the robot in can be anywhere on the workspace.

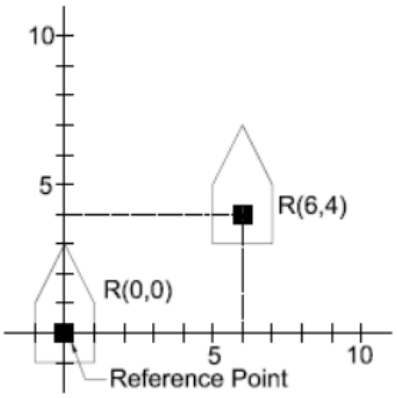


Figure 3.9 : Work space and translational factor of the robot.

In another case the robot is able to change its orientation by rotating around its reference point. Then we have another factor to describe the position of the robot as an angle of the robot from the x axis. This angle is named φ and addes another degree of freedom for the robot beside translational factor x and y. We now define the $\mathcal{R}(x, y, \varphi)$ for the robot that describes the location and orientation of the robot in 2-dimentional space. The angle φ, is the clockwise rotational of the robot that is specified initially at $\mathcal{R}(0,0,0)$. Considering the rotation of 45 degrees of the robot aroun its refrence point the place of the robot will be described as $\mathcal{R}(6,4,45)$ as in figure 3.10. Briefly, the placement of the robot is describe by three parameters and this number of parameter corresponds to the number of *degrees of freedom* of the robot (DOF). In a 2-D space, there are three degrees of freedom. However, in 3-dimensional environments there are six degrees of freedom. Namely, three of them describe the position of the robot in (x, y, z) space and the other three describes the rotational elements of the robot in pitch, yaw and roll form. So the placement of the robot is shows as $\mathcal{R}(x, y, z, \varphi, \theta, \beta)$ and has six degrees of freedom.

This parameter space of the robot $\mathcal{R}$ is called *configuration space* and is shown by $\mathcal{C}(\mathcal{R})$ and the point p corresponds to the point of robot in this configuration space.

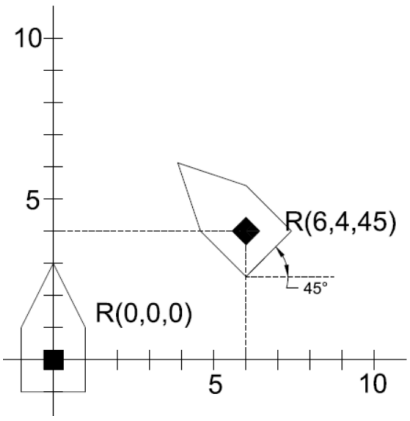


Figure 3.10 : Rotational factor of the robot.

So, the work space is the space in which the robot moves around in the real world and the configuration space is the parameter space of the robot. Now we are able to specify a placement for the robot by specifying a point in the configuration space.

Although we consider the case of point robot, it is useful to know how it is different for robots to differ between work space and configuration space. If we consider our robot as a point robot then the robot can move freely in our space by defining the shortest path and will never collide the obstacles. However, if our robot is a polygon robot, and we have considered a point in the robot as a reference point, then there is a change that the robot collides with obstacles in this path. These points are the shaded area in figure 3.11 and are relative to reference point of robot. The configuration space is where the robot is able to move on the path that are the white regions in the graph and does not consist of black and grey parts. Nevertheless, the work space is the environment which consists of these grey areas, not black ones, and is the space in which the polygonal robot moves around to reach to the destination [22]

3.4.2 Definition and algorithm

Before we plan the trajectory for the mobile polygonal robot. We describe some space configuration for the robot to make this path more accurate and simple to find. As we discussed before, we denoted robot by $\mathcal{R}$ and obstacles by $\mathcal{P}_1, \ldots., \mathcal{P}_N$. The obstacles are polygons with the total number of vertexes denoted by n. To simplify the environement of the robot and obstacles we restrict motion of the robot to a large bounding box B that contains all obstacles in it. The free configuration space $\mathcal{C}_{free}$ is now consists of the part of B that does not cover obstacles as in figure 3.12.

$$\mathcal{C}_{free} = B \setminus \bigcup_{i=1}^{N} P \tag{3.5}$$

The free space is a disconnected region that may have holes in it and the main goal is to compute a free space in a way that the robot will be able to find a path from start node to the goal node. For this purpose, we use *trapezoidal map*.

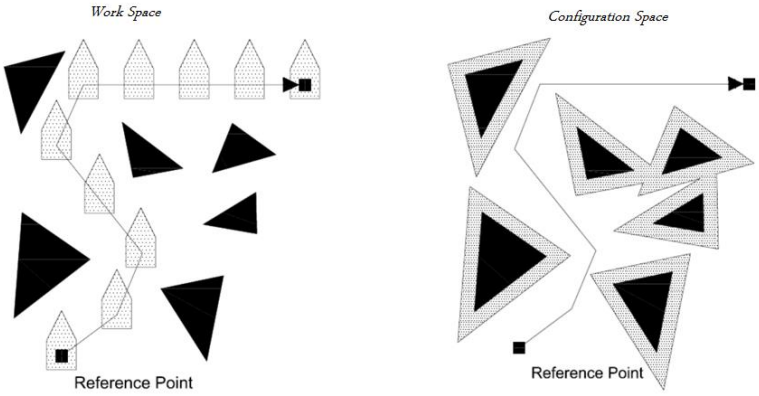


Figure 3.11 : A patch in the work space and its corresponding curve in the configuration space.

Definition

Trapezoidal map is a set of non-intersecting and parallel lines that are achieved by drawing vertical lines from the each vertex of obstacles upward and downward. These lines will stop when they intersect with the line of boundary or any other obstacle.

The algorithm of computing free space using trapezoidal map is described below:

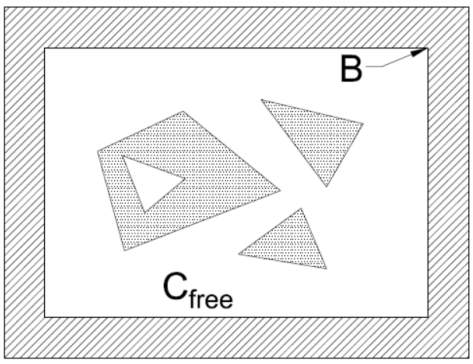


Figure 3.12 : Free space and unlimited boundary B.

Algorithm:

i. Consider a set of S disjoint polygons as obstacles as an input.

ii. Consider E the set of all edges of polygons in S.

iii. Compute Trapezoidal map $\mathcal{T}(E)$ with its related method described above in the definition of Trapezodial map.

iv. Remove all the lines that are made inside the obstacles.

v. The output of the algorithm is a trapezoidal map of $C_{free}(\mathcal{R}, S)$ for a point robot $\mathcal{R}$.

The algorithm is illustrated in figure 3.13. Part a, describes third step of algorithm which we produces the lines attached to the vertex of the polygon obstacle and continues until it touches a surface. Part b, shows the graph in which the trapezoids inside the obstacle are removed and ready.

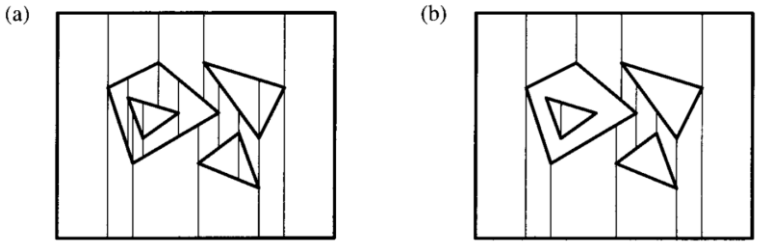


Figure 3.13 : Trapezoidalized space.

Let us consider the above graphs containing obstacles to route from one distinct point to another in any place inside $\mathcal{C}_{free}$. The trapozides in figure 3.13-b divides the workspace into some sub-regions. The starting and ending point of the robot`s motion can be anywhere in these trapezoids. If the starting node,p_{start}, and the destination node,p_{goal}, are in different trapezoids so the path of robot will pass through a number of trapezoids and it may have to turn in some of them. Actually, by using these trapezoids and guiding robot through them we construct a road map that helps the robot reach its destination from the start point to the goal point. The road map is a

graph, $\mathcal{G}_{road}$, which is located in the free space and paths will always follow this road map.

Let us define the road map as follows; as far as every neighbor trapezoids share a vertical edge together, we place a node at the center each edge. We also place a node at the center of each trapezoid section. An arc is sketched between each neighbor trapezoids and line segments from the center of trapezoid to the center of neighbor line segments. Continuing this procedure until we reach to the final trapezoid that consists of the goal point. At this point, we have a road map constructed. Figure 3.15 illustrates it.

We use the road map and trapezoidal map to plan a short motion from start point to end. For this end we first determine trapezoids Δ_{start} and Δ_{goal} that consist these points. If both of these nodes are in a same trapezoide then the road map will be a straight line, if not, let v_{start} and v_{goal} be the center points of these trapezoids. Therefore, the path will consist of three part: the first one is from p_{start} to v_{start}, the second is the path between v_{start} and v_{goal}, and the last part is the straight line between v_{goal} andp_{goal} that is illustrated in figure 3.16.

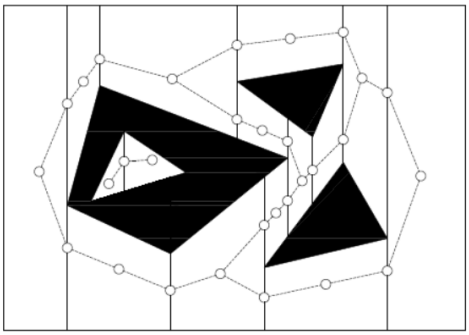


Figure 3.14 : A road map.

The algorithm for computing the path can be described in steps below:

Algorithm:

i. Determine definitions the road map $\mathcal{G}_{road}$, start position p_{start}, goal position p_{goal} and trapezoidal map $\mathcal{T}(\mathcal{C}_{free})$.

ii. Find trapezoids that consist of the points p_{start} and p_{goal}.

iii. Let v_{start} and v_{goal} be the center points of the Δ_{start} and Δ_{goal} respectively.

iv. Find the straight line from p_{start} to v_{start} and from v_{goal} to p_{goal}.

v. Compute a path from $v_{start}\ to\ v_{goal}$.

vi. If there is no such a straight line start node and goal node are at the same trapezoid, else the path is calculated

3.4.3 Shortest path for a point robot

In the previous section after finding start and goal trapezoids, we find a road map between the nodes of the center of these trapezoids by the use of breadth-first search[7]. This method uses the minimum number of arcs in the road map but this is not necessarily a short path. In a way by giving a weight to each arc, we can find a shortest path in the road map. An algorithm that uses weighted arcs to find the shortest path can be Dijkstra`s algorithm. Although a short path can be found using this method but it does not mean necessarily that the shortest path is found. It is illustrated in figure 3.15 that a short path from p_start to p_goal in the road map passes below the triangle obstacle but the real short path passes above it. Therefore, what we need is another road map which guarantees the short path that we find in the road map is really the shortest path of all.

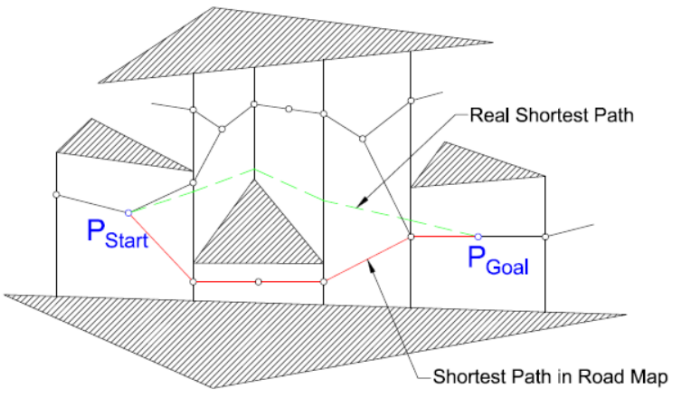


Figure 3.15 : A short path on the road map and the real shortest path.

The shortest path between p_{start} and p_{goal} is a polygonal path which its inner vertices are the vertices of S disjoint polygonl obstacles (figure 3.16). The inner vertex of the

shortest may “a” is the vertex of the obstacle, this is also correct for the “b” and “c” vertices. It is seen the inner vertices of the road map are the vertices of the obstacles.

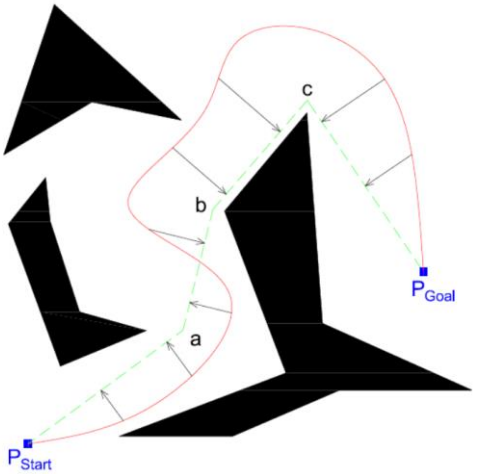


Figure 3.16 : Inner vertices of the shortest map a, b and c.

3.4.4 Computing the visibility graph

With this characterization of the shortest map and its structure we are now able to construct a short map that lets us find the shortest path named visibility graph of S, $\mathcal{G}_{vis}(S)$. As it is mentioned before its nodes are vertices of the obstacle and there is an arc between two vertices v and w that can see each other which is called visible and the segment connecting these tow vertices to each other is named visibility edge and is denoted by $\overline{vw}$. It is obvious that all the vertices of any obstacle S, see each other, so the edges of obstacle are also subsets of $\mathcal{G}_{vis}(S)$.

Let us include the start and goal nodes to the vertices of the visibility graph of the set $S^* \coloneqq S \cup \{p_{start}, p_{goal}\}$, so the arcs that are made out of these nodes will be the visible arcs. Therefore, $\mathcal{G}_{vis}(S)$ is included of arcs between verteces, which also include the start and goal node, which are visible to each other or in other words see each other.

The algorithm of finding the shortest path using visibility graph methods as shown in figure 3.18 can be described as following steps:

Algorithm:

i. The input is the set of disjoint polygonal obstacle and tqo points p_{start} and p_{goal}.

ii. The visibility graph is denoted by $\mathcal{G}_{vis}(S)$ and is defined as $S \cup \{p_{start}, p_{goal}\}$.

iii. Assign a weight for each arc(v, w) in $\mathcal{G}_{vis}(S)$ which is the Euclidean length of the segment $\overline{vw}$.

iv. Use Dijkstra`s algorithm to find the shortest path between p_{start} and p_{goal} using the weighted arcs.

v. The shortest collision-free path is gained.

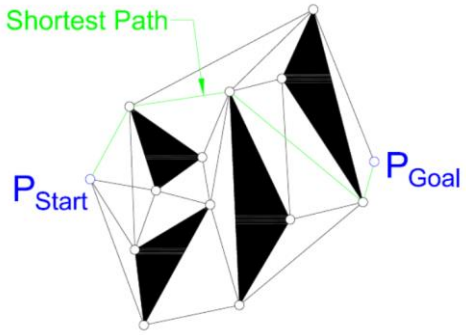


Figure 3.17 : The short path in the road map using Djikstra`s method

3.5 Decomposition Cell Method

Beside previous method in path planning, this method in much more practical and solves motion planning problems in all dimensions of obstacles in free space, say 2-dimensional or 3-dimensional. The aim of this method is to decompose the robot`s free space,$\mathcal{C}_{free}$, into a number of non-overlapping sections, called cells which their set can be expressed as $\{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_i, \dots, \mathcal{C}_N\}$ which $\mathcal{C}_i \epsilon \mathcal{C}_{free}$ and the union of $\mathcal{C}_i`s$ generate $\mathcal{C}_{free}$. Next, a graph is made using the gravity center of these cells and borderlines of each cell. After that, an optimal path will be found using some of algorithms such as Dijkstra`s algorithm described in previous sections. Finally, a path is extracted from these sequences. There are two methods that are going to be explained here *triangulation decomposition* and *cylindrical decomposition.*

Triangular decomposition

Consider a 2-dimentional space with C-obstacles that form a polygonal region in $\mathcal{C}$. For the simplicity we imagine $\mathcal{C}$ is bounded by $\mathcal{B}$. We define triangular decomposition of free space $\mathcal{C}_{free}$ by connecting each node in space, consisting start and goal points and the vertex of obstacles that sea each other.

The connectivity graph G will be established by connecting two nodes in G for the cells that have a common arc in other words two cells that are adjacent. Figure 3.19 shows convex polygonal decomposition of the robot`s free space whose interiors do not intersect and the connectivity graph related to it. Each cell is labeled with a numbered the associated connectivity graph is displayed with bold lines by connecting the center of gravity point of each cell.

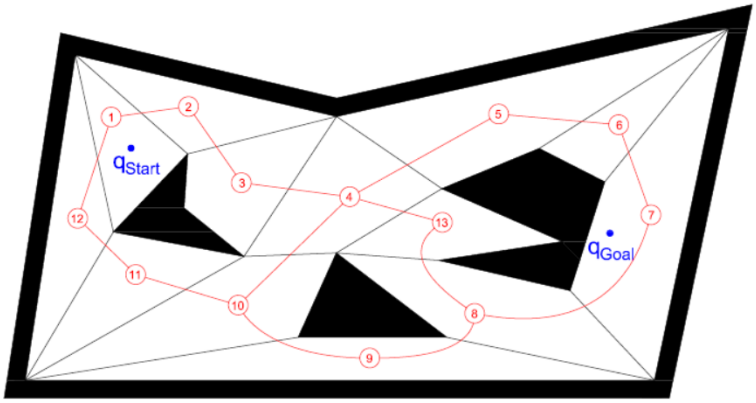


Figure 3.18 : Triangulation decomposition with non-overlapping cells which produces the connectivity graph

Note that there are many ways to triangulate $\mathcal{C}_{free}$, but the real problem is to find good triangles which are not thin is of a considerable attention in computing the geometry. A good approach can be gain by incrementally split faces of triangles by attempting to connect tow vertices of a face by an arc. If this line segment does not overlap other segments then a right arc has been drawn for decomposition purpose. The output of the algorithm is a sequence of cells $(k_1, \dots, k_p)$ which is called channel. This channel consists of the initial point located in the first cell $q_{start} \epsilon\, k_1$ and destination point located in the last cell $q_{goal} \epsilon\, k_p$ and for every $j\epsilon\, [1, p-1]$, k_j and k_{j+1} are adjacent.

For example, in figure 3.19 q_{start} is in cell 1 and q_{goal} is in cell 7. Therefore, this channel is consisted of set of cells{1,2,3,4,13,8,7}. The interior of the channel which is:

$$nt\left(\bigcup_{j=1}^{p} k_j\right)$$

Lies entirely in $\mathcal{C}_{free}$.

Now that we have decomposited the whole space, one way to find a road map is to connect every point that is located in the middle of each line segment of the adjacent cells named $\beta_j = \partial k_j \cap \partial k_{j+1}$, where ∂k_j denoes the boundary of the cell k_j. These point are named by Q_i , $i = 1,2, \dots, p-1$. As in figure 3.20, By connecting initial point to the center of first arc Q_1 and continuing the method as told before, we will reach to the last center of line segment Q_6. Connecting this point to the goal point will lead us the shortest road map possible.

Therefore, once a channel has been constructed, a free path can be generated by joining q_{start} and q_{goal} through the midpoints of the arcs of boundary shared by every two successive cells in the channel.

3.6 Probabilistic Road Map (PRM)

The previously mentioned methods include high computational cost of the distance transform, therefore makes them infeasible for large maps. Probabilistic methods method, on the other hand, due to their structure and the method of finding a path between initial point and the goal point, is more of a use and the most well known and useful of them is Probabilistic road map (PRM) method. Creating a feasible path for the robot has two steps, first planning, and second query. In the first step $\boldsymbol{N}$ random points are found on the free space $\boldsymbol{\mathcal{C}}$, and the number of these points is optional. Each of the points is connected to its neighbors by a straight line that does not cross through any obstacle. Therefore, a graph of network with a minimum number of disjoint components with no cycles is created. The advantage of this method is that few points in the free space will be tested to determine a path between them with no obstacle. As

shown in figure 3.19 the dots represent the randomly selected points and the grey lines are obstacle free paths between those points.

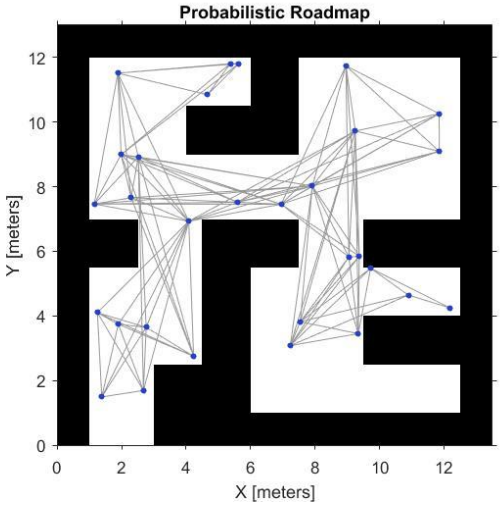


Figure 3.19 : Nodes found from PRM path planning method

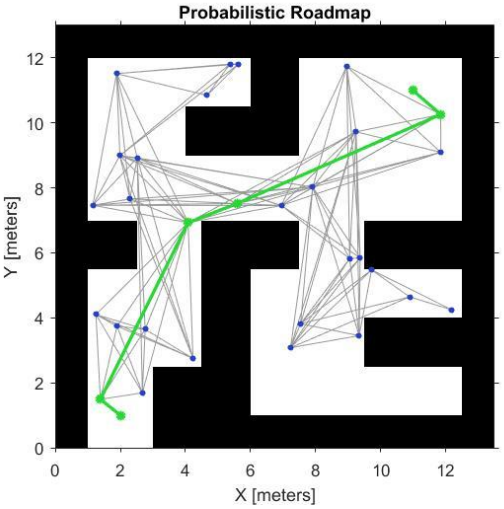


Figure 3.20 : Road map found with PRM method

The second step is finding a path from the start point to the goal. This is simply possible with moving to the closest node in the roadmap and iterating this step until getting off at the node closest to the goal point. The first starting point and the goal point will also be included in the roadmap after finding the path. Finally, as in figure 3.20, the road map is found ant it is shown as green line.

4. PATH TRACKING METHODS

Path tracking is process concerned on how to determine speed and steering configurations at each instant of time for the robot, using the positional information obtained from GPS,IM and vehicle odometery, in order for the robot to follow a desired path and reach to the goal point. The robot is able to follow a certain path if and only if there is continuity in the position orientation and curvature of the path that are satisfied by path generator. In principle, a path can be considered as a continuous function but in reality, it is discretized and described in a series of straight lines that are connected through the waypoints in the path. The waypoints are series of the points that are described in the path in order for the robot to be able to follow the path accurately. There are many algorithms to track a path. These algorithms are described in the following sections as follow-the-carrot, pure pursuit and vector pursuit.

4.1 Follow the Carrot

This algorithm is based on obtaining the goal point and aiming the robot directly toward that point. As it is obvious in figure 4.1, there is a difference between the robot`s place and the path. A line is drawn from the center of robot perpendicular to the trajectory to be followed. The distance from this point to the goal point is called the look-ahead distance .The intersection of robot`s path and look-ahead distance is the carrot point or goal point [23].

The most important parameter here is the orientation error e_0, which is the angle between robot`s current heading direction and the line drawn from the center of robot to the carrot point. If robot is heading exactally to the goal point, then the error will be zero. Therefore, for minimizing the orientation error, a proportional control law is considered and the magnitude of a turn φ is driven from equation 4.1 where k_p is the proportional gain and e_0 is the orientation error.

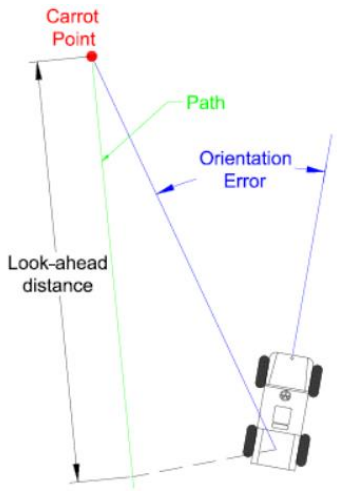


Figure 4.1 : Follow-the-carrot.

$$\varphi = k_p e_0 \tag{4.3}$$

Adding derivative or integrative functions, it is possible to increase the accuracy of the controller. In this method the vehicle is always very close to the trajectory so the goal point is found by adding a particular value to the current x-coordinate and then finding the corresponding y-coordinate. So the steps for the robot to follow the path can be said as below:

i. Find the goal point

ii. Turn toward desired direction

iii. Move straight toward the small look-ahead distance which is found before.

iv. Find the current location

v. Find the nest goal point by a small look-ahead distance and repeat the algorithm.

Although this method is simple to implement and easy to understand, there are some drawbacks. Firstly, as robot tries to turn immediately toward each new carrot point, it cuts the corners and does not follow it exactly. Secondly, the vehicle can oscillate on the way to the goal point due to high speed of a small look-ahead distance. This methodology in path tracking is not an accurate one, so it cannot be used in the projects that need to follow the path accurately. Still it is useful for educational purpose and to comparison with other methods.

The position tracking of the vehicle is done continuously every time the vehicle moves

$$y_c = y_l + d` \sin(\theta_c) \tag{4.2}$$

straight for a desired length of look-ahead distance or every time the vehicle turns and its orientation changes. The vehicle starts at the particular position with particular coordinates and heads toward the goal point with particular angle. The angle is measured by using magnetometer and the distance moved is calculated by the means of rotary encoders. The initial starting angle is stored in the vehicle by magnetometer and whenever the vehicle turns, it means its angle changes, the current heading of the vehicle is calculated by subtracting the initial angle from the current reading of the magnetometer.

Suppose that the current coordinates of the robot are (x_c, y_c), and the position of the robot which was the last time calculates to be (x_l, y_l). Let d` be the distance traveled by robot and θ_c to be current heading angle (figure 4.2). Then we will have equations 4.2 and 4.3.

$$x_c = x_l + d \cos(\theta_c) \tag{4.3}$$

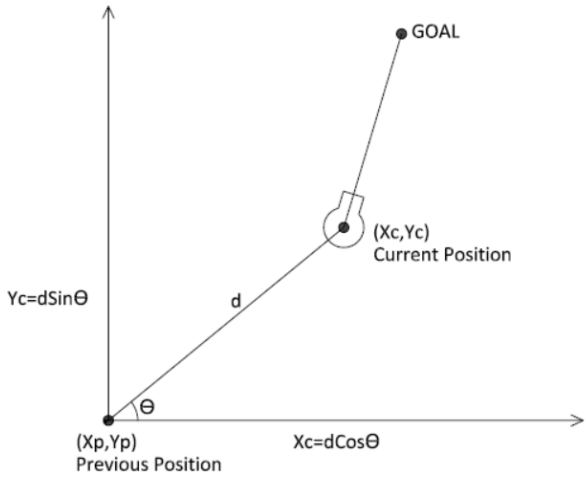


Figure 4.2 : Current and previous location of robot and the relation in-between.

Using the above equations, the place of robot can be calculated in each step.

4.2 Pure Pursuit

The pure pursuit method is devised to compute the curvature arc necessary for the robot to be able to go from the current position to some goal position. This goal positions slide smoothly through the path as shown in figure 4.3. The goal waypoint $n + 1$ in each step is chosen in a way to bring the vehicle back to the way. The name of this algorithm, pure pursuit, comes from the analogy that is used in this method that is chasing a point some distance ahead of it which is a look-ahead distance,l, far away from it(lookahead points 1,2 and 3).

It can be compared by the way human being drives a car. We tend to look some distance ahead of the car and head toward that point. In other words in pure pursuit method the vehicle pursues the points that are a look ahead distance away from the current location. These iterations continue until finally the vehicle reaches its destination.

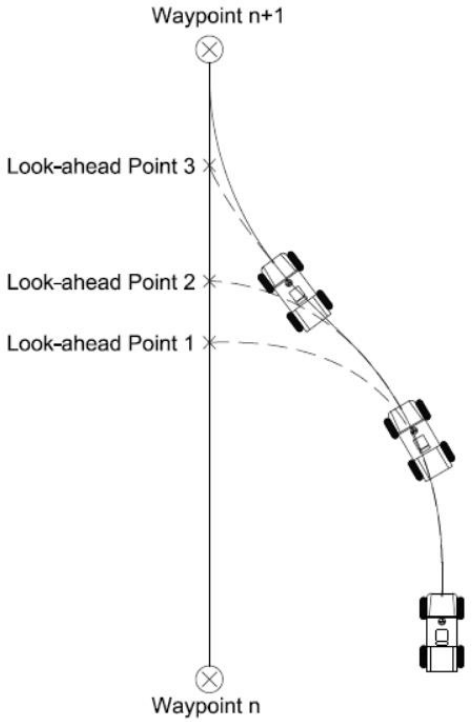


Figure 4.3 : Sequential look ahead points on a straight line path between two waypoints.

Mathematical definition

Let us define the position of look ahead distance by (x_g, y_g) relative to the vehicle and length of look ahead distance L. The lookahed point with coordinates (x_g, y_g), is lookahead distance away from the vehicle, on the path to be followed. Let θ_{error} be the angle difference between look ahead point and direction that robot heads to. In pure pursuit method there will be a curvature arc for robot to follow,γ, and a radius of r which is the radius of the circle which the arc γ is part of it. The radius r is the distance from the origin of robot to the origin of instantaneous center of rotation (ICR). Therefore γ, is the inverse of the distance r (figure 4.4). It has a sign, which is negative when the vehicle rotates clockwise and positive when it turns counterclockwise.

Therefore, from Pythagoras theorem, we have equation (4.4) and (4.5).

$$x_g{}^2 + y_g{}^2 = L^2 \tag{4.4}$$

$$d^2 + y_g{}^2 = r^2 \tag{4.5}$$

From figure 4.4 it is obvious that $d = x_g - r$. Substituting d in equation 4.5 leads to equation (4.6).

$$\begin{aligned} \left(x_g - r\right)^2 + y_g{}^2 &= r^2 \\ x_g{}^2 + y_g{}^2 &= 2rx_g \end{aligned} \tag{4.6}$$

Substituting equation 4.6 into equation 4.4 leads to

$$\begin{aligned} 2rx_g &= L^2 \\ r &= \frac{L^2}{2x_g} \end{aligned} \tag{4.7}$$

And we have

$$\gamma = \frac{1}{r} \Rightarrow \gamma = \frac{2x_g}{L^2} \tag{4.8}$$

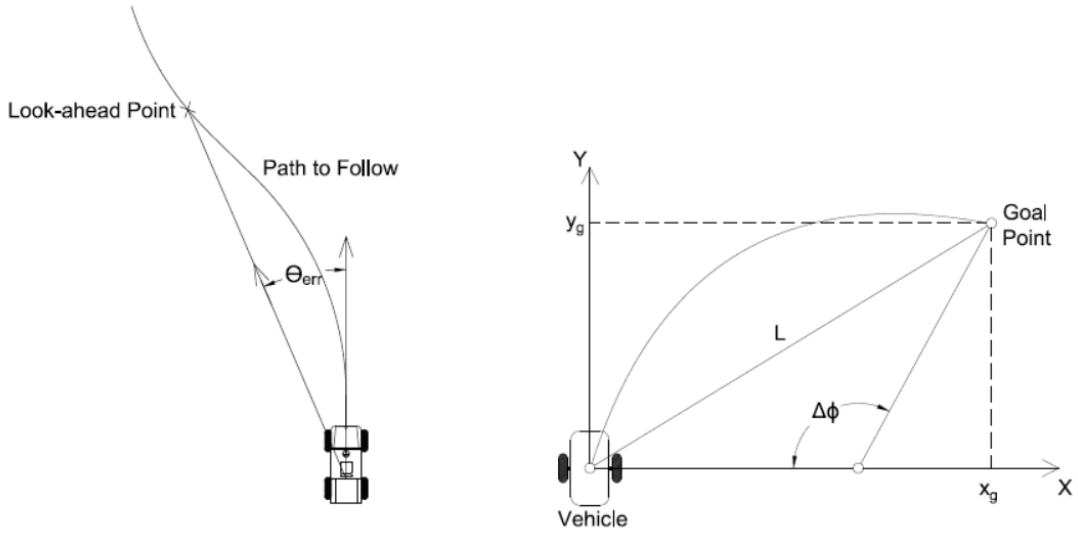


Figure 4.4 : The left graph shows the geometry of pure pursuit method and the path to be tracked. The right graphs shows the curve to be calculated in pure pursuit method.

As it was previously mentioned in equation 4.1, the pure pursuit algorithm, also, is a proportional controller with θ_{error} as a parameter to be controlled a gain.

Due to figure 4.1-left

$$sin(\theta_{error}) = \frac{x}{L} \tag{4.9}$$

For small angles it is said that $\sin\beta = \beta$, therefore, for small heading errors $\sin(\theta_{error}) = \theta_{error} \cong \frac{x}{L}$. Substituting this result in equation 4.8 results in the control law γ:

$$\gamma = \frac{2}{L}\,\theta_{error} \tag{4.10}$$

As it is shown in figure 4.1-right, the curvature also represents the instantaneous change of orientation of the robot $d\varphi$, with respect to traveled distance ds as follows:

$$\gamma = \frac{1}{r} = \frac{d\varphi}{ds} \tag{4.11}$$

In pure pursuit strategy, in every control interval T_0,the robot changes its curvature set-point γ_{sp} by fitting an arc, that is tangent to axis Y, to a destination point in the path at a certain look ahead distance L. Assuming $\Delta\varphi$ as the heading change of robot along the curvature arc, the goal point (x_g, y_g) can be computed as equation (4.12) [24].

$$\begin{aligned} x_g &= \frac{\cos(\Delta\varphi) - 1}{\gamma_{sp}} \\ y_g &= \frac{\sin(\Delta\varphi)}{\gamma_{sp}} \end{aligned} \tag{4.12}$$

Using equation 4.4 and 4.12 results in:

$$L^2 = \frac{2 - 2\cos(\Delta\varphi)}{\gamma_{sp}^2} = -\frac{2\,x_g}{\gamma_{sp}} \tag{4.13}$$

Therefore, the curvature control law is obtained as below:

$$\gamma_{sp} = -\frac{2}{L^2} x_g \tag{4.14}$$

For small orientation errors $sin\theta_{error} = \theta_{error} = \frac{x_g}{L}$.

So we have:

$$\gamma_{sp} = -\frac{2}{L^2} x_g = \frac{2}{L} \theta_{error} \tag{4.14}$$

As it is concluded in equations 4.10 and 4.14, the algorithm has a gain of control law, $2/L$, and an error signal, θ_{error}, in small orientation errors, and a gain of $2/L^2$ and error signal,x_g, in big orientation errors. So, it has only one single parameter actually, the look ahead distance L and this makes the algorithm easy to implement. A longer look ahead distance results in smaller gain so steering control will be smoother but a drawback is that the tracking accuracy will reduce. On the other hand, a shorter look ahead distance results in larger gain, so the steering command increases and the vehicle`s motion may be unstable, but an advantage can be mentioned as reduced tracking errors.

4.3 Vector Pursuit

Sir Robert S.Ball first introduced this method in 1900 and is a new path tracking method based on screw theory. The method is developed for applications in kinematics and statics of rigid body and provides a mathematical formulation for the geometry of lines which are central to rigid body dynamics. Any motion of the rigid body can be described as a rotation about a line in space with an associated pitch. Previous methods focused on tracking path and getting to the goal point and did not consider the orientation of look ahead distance, whereas this method works on getting the vehicle to the goal point in a current orientation and curvature.

As screw theory, a screw consists of a centerline and a pitch that are defined in a given coordinate system. Every centerline, also, can be represent by Plucker line coordinates that is representing a line using only two points on the line. These points represent by vectors r_1 and r_2. This centerline can be defined as a unit vector S, which is **shown using two parameters: the direction of the line and a moment vector S_0 of the** line about the origin (Figure 4.5).

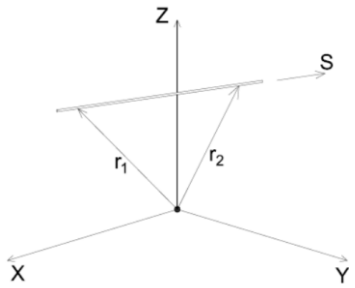


Figure 4.5 : Line S defined by two points represented as vectors r_1 and r_2.

Figure 4.5 demonstrates that $\boldsymbol{S}$ and $\boldsymbol{S_0}$ can be represented as in equations (4.15) and (4.16) respectively.

$$S = \frac{r_2 - r_1}{|r_2 - r_1|} \tag{4.15}$$

And

$$S_0 = r \times \underline{S} \tag{4.16}$$

It can be seen from equations above that the components of S, are directions of the line and are dimensionless, and the components of S_0, the moment of line have units of length. The Plucker line coordinates of this line can be shown by vectors $(S; S_0)$. By defining parameters below;

$$S = [L, M, N]^T, \qquad S_0 = [P, Q, R]^T, \qquad r_1 = [x_1, y_1, z_1]^T, \qquad r_2 = [x_2, y_2, z_2]^T$$

It is mathematically understood that

$$\begin{aligned} L &= \frac{x_2 - x_1}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}} \\ M &= \frac{y_2 - y_1}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}} \\ N &= \frac{z_2 - z_1}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}} \end{aligned} \tag{4.17}$$

and

$$\begin{aligned} P &= y_1 N - z_1 M, \\ Q &= z_1 L - x_1 N, \\ R &= x_1 M - y_1 L. \end{aligned} \tag{4.18}$$

Figure 4.6 shows the instantaneous motion of a rigid body that rotates around screw,$\bar{\bar{S}}$, with an angular velocity,ω, that has a centerlined as defined before as $(S; S_0)$, that has a pitch h. The pitch of screw is shown in figure 4.6. The velocity of any distinct point

on the rigid body is the sum of angular velocity and translational velocity due to the pitch of the screw. Actually pure pursuit method computes two instantaneous screws. The first screw represents the translational screw, $\bar{\bar{S}}_t$, which is the translation from the current location of a rigid body to the look ahead point. The second screw, that is rotational screw $\bar{\bar{S}}_r$, accounts for rotation from the current orientation of a rigid body to the orientation at the look ahead distance These two values are added together to gain an instantaneous motion of the vehicle. If we define r as any vector from the origin to the centerline of scre we will have the velocity of rigid body as in equation (4.19).

$$\omega\, \bar{\bar{S}} = (\omega\, S;\ \omega\, S_{0h}), \tag{4.19}$$

Where

$$S_{0h} = S_0 + hS = r \times S + hS, \tag{4.20}$$

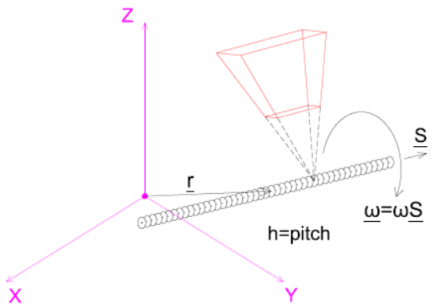


Figure 4.6 : Instantaneous motion about a screw.

There are two exclusive cases for the screw of rigid body. Case one, translation screw: The motion about screw with an infinite pitch results in pure translation of a rigid body with velocity v along the direction S [25]. Therefore, equation 4.19 simplifies to equation 4.21, which has a centerline at infinity.

$$v\, \bar{\bar{S}} = (0; v\, S) \tag{4.21}$$

Case two, rotational screw: on the other hand, the rotation around screw which pitch is equal to zero results in a pure rotational motion of a rigid body. Therefore, equation 4.19 simplifies to equation 4.22 by substituting a pitch h, equal to zero.

$$\omega \bar{\bar{S}} = (\omega S; \omega S_0) \tag{4.22}$$

Although equation 4.12 is a translational screw and equation 4.22 is a rotational screw, but they have different units, i.e., the first three components have units of rad/time and last three components have units length/time

Then, the screws will be calculated depending on whether the nonholonomic constraints of the rigid body are initially ignored or counted in the calculations. These screws, as mentioned before, are translation and rotation screws and then they are sum up with each other as in equation 4.23 to form the desired instantaneous motion of the

$$\bar{\bar{S}} = \bar{\bar{S}}_t + \bar{\bar{S}}_r \tag{4.23}$$

robot. All these information are further used to calculate the desired turning radius of the robot from its current radius in current position to the end position [16].

The initial constraint that appears here is that the location of centerline of instantaneous screw must lie on the vehicle's y-axis. Hence $\bar{\bar{S}}_t$ in world`s coordinate system will be defined to be

$$w_{\bar{\bar{S}}_t} = k_t(0,0,1; w_{y_v} + \frac{d^2}{2\, v_{y_L}} \cos\theta_v\,, w_{x_v} + \frac{d^2}{2\, v_{y_L}} \sin\theta_v\,, 0) \tag{4.24}$$

Where,

d : Distance from the origin of the vehicle`s coordinate system to the origin of the look-ahead coordinate system

(v_{x_L}, v_{y_L}) : Coordinates of the look-ahead coordinate system`s origin in the vehicle`s coordinate system

(w_{x_v}, w_{y_v}) : Coordinates of the vehicle`s position in the world coordinate system

θ_v : Angle from the x-axis of the world coordinate system to the x-axis of the vehicle`s coordinate system

k_t : Weighting factor

Equation 4.24 is valid if the term $v_{y_L} \neq 0$, otherwise $\bar{\bar{S}}_t$ will be:

$$w_{\bar{\bar{S}}_t} = k_t(0,0,0;\frac{w_{x_L} - w_{x_v}}{d}, \frac{w_{y_L} - w_{y_v}}{d}, 0) \tag{4.25}$$

And instantaneous screw $\bar{\bar{S}}_r$ will be:

$$w_{\bar{\bar{S}}_r} = k_r(0,0,1; w_{y_v}, -w_{x_v}, 0) \tag{4.26}$$

and

$$w_{\bar{\bar{S}}_d} = w_{\bar{\bar{S}}_r} + w_{\bar{\bar{S}}_t} \tag{4.27}$$

The amount of rotation, φ, can be defined by (figure 4.7),

$$\varphi = atan2((2\,{v_{y_L}}^2 - d^2), (2\,v_{x_L} v_{y_L}) - atan2(\frac{d^2}{2\,v_{y_L}}, 0)) \tag{4.28}$$

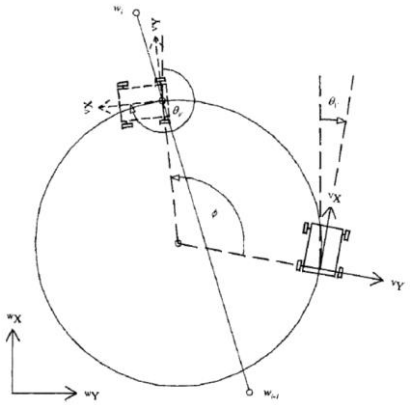


Figure 4.7 : Rotation defined by screw theory

Where the value of φ mus be in the interval $(0,2\pi]$ rad. It is noted that the last part of equation (4.28) will always be $\pm {}^{\pi}/_{2}$ radian and it depends only on the sign of v_{y_L}.

5. SIMULATION AND RESULTS

Using simulation environments it is easy to show how a differential robot that is the case of our study works in different kinds of path tracking methods. Using MATLAB workspace for simulation and extracting output of pure pursuit and vector pursuit methodologies is what is covered in chapter five of this thesis. There is a sub section in MATLAB Toolbox named *Robotics system toolbox* that is presented in year 2015 and has simulation concepts for robotic science.

5.1 Path Information

Before taking any step in simulations of mobile robot`s path tracking, firstly a path should be planned. Using MATLAB simulations a desire path using a robot motion model is produced. The trajectory from start point to goal point of robot`s path is named road map. A Probabilistic Roadmap (PRM) planner constructs a road map for a given free space environment of robot, using randomly sampled nodes in this space and connecting them to each other. When the roadmap is constructed now it is possible to ask for a path from initial point of robot to its destination.

Another important fact that have to be considered is the dimensions of the robot. If its dimensions are not considered in path following procedure then it may have collisions with obstacles or walls in its path. Therefore, the dimensions of the map should be inflated with respect to the dimension of robot. Considering the size of robot and inflating the map due to it, it is now possible to construct an obstacle free map.

Using related MATLAB codes a free space $\mathcal{C}_{free}$, is constructed with the size 41×52 meters as shown in figure 5.1. The map is represented as an occupancy grid map using imported binary data. PRM uses this binary occupancy grid representation to deduce free space, when sampling nodes in free space of a map.

In this map the dimensions of the robot is not considered yet. This may result, after finding a road map using PRM method, in collision of robot`s body with the walls in the environment. Therefore, a consideration of robot`s size should be implemented

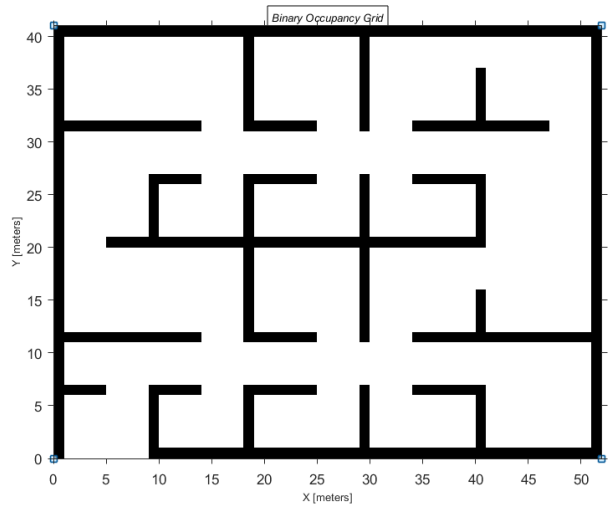


Figure 5.1 : A 41 × 54 meters dimension map.

into the system. Suppose our robot as a differential and circular robot with radius of 0.3. The new free space is now collision free and is safe for the rigid body to move all along the road (Figure 5.2).

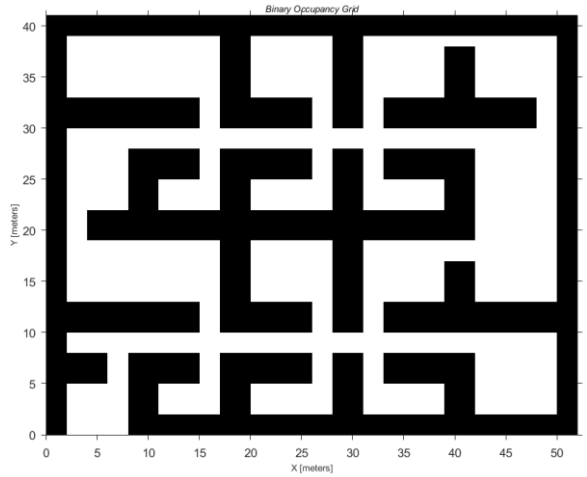


Figure 5.2 : Inflated map due to the dimensions of robot.

Writing new codes updates PRM object with newly inflated map and sets parameters and define other attributes. The numbers of nodes (waypoints) and maximum allowed distance between them also can be selected willingly. The large maximum distance can increase the connectivity between two nodes to find a path easier, on the other hand, it can also increase the computation time to create a road map. In figure 5.2, 50 nodes are selected to be in the map. The visible nodes that can see each other make a connection in between.

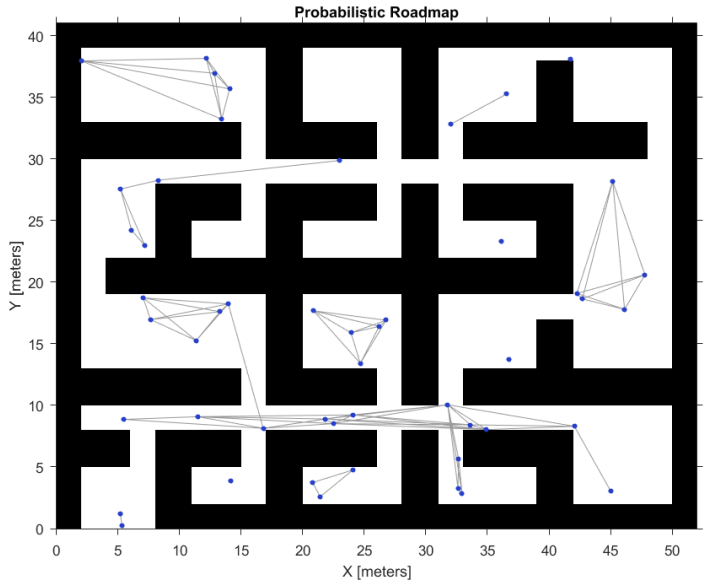


Figure 5.3 : Selected waypoints in the map using PRM and connection between visable nodes

Then, giving the coordinates of the start location and goal location and using PRM with respect to MATLAB codes will lead to a road map for our differential robot. The coordinate`s information of each node that is traversed in the map and is on the road map, is also known. Finally, the road map is in hand (Figure 5.4). The coordinates of nodes from start location to destination are:

path =

45.0000 37.0000

46.8723 36.7622

49.4179 36.0306

48.8283 25.6962

37.2370 29.8936

24.9039 29.4216

6.2186 28.6511

2.5351 18.1202

16.3754 13.7021

15.4667 7.7874

15.0000 4.0000

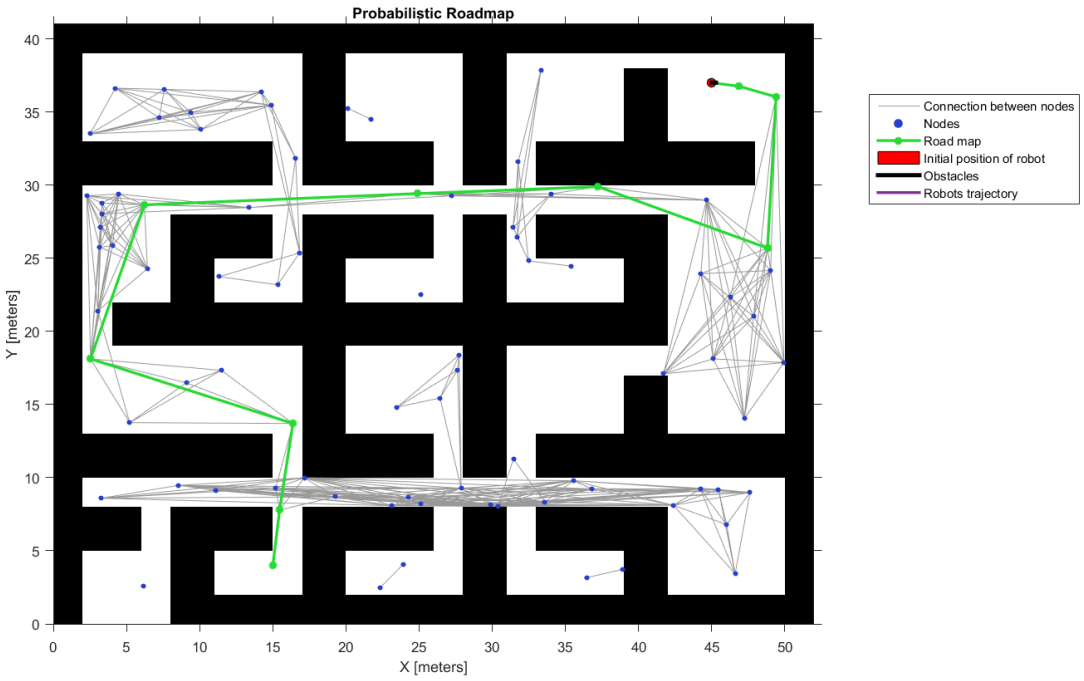


Figure 5.4 : Probablsitic road map of the robot to be surveyed.

5.2 Path Tracking

In this part the methods of path tracking that had been introduced in chapter 4 will be implemented on the robot in its`s road map which is found in section 5.1. Among these methods the simulation results of the pure pursuit and vector pursuit methods have been covered here and a comparison is made between two of them.

There are some factors tht have to be defined in implementing the algorithm in MATLAB environment. The linear velocity of robot is a constant value all over the path, So it can be changed to a desirev value at any point. The angular velocity command is a velocity that moves the robot from its current position to reach some look-ahead point in front of robot. To imagine this fact more feasible, think of a robot constantly chasing a point in front of it. In Robotics System Toolbox by using a controller we specify a list of waypoints . It is also inportant to have an information of the refrence coordinate system and its inputs and outputs. As it is shown in Figure 5.5 there are six waypoints on the road map and positive x and y directions are in the right and up directions respectively (blue in figure is shown), also the angle difference between orientation of the robot measured counterclockwise in radians from positive x axis is named by θ. The robot`s pose (position and orientation) is input for the system in MATLAB as [x y teta]. The path following controller provides input control signals

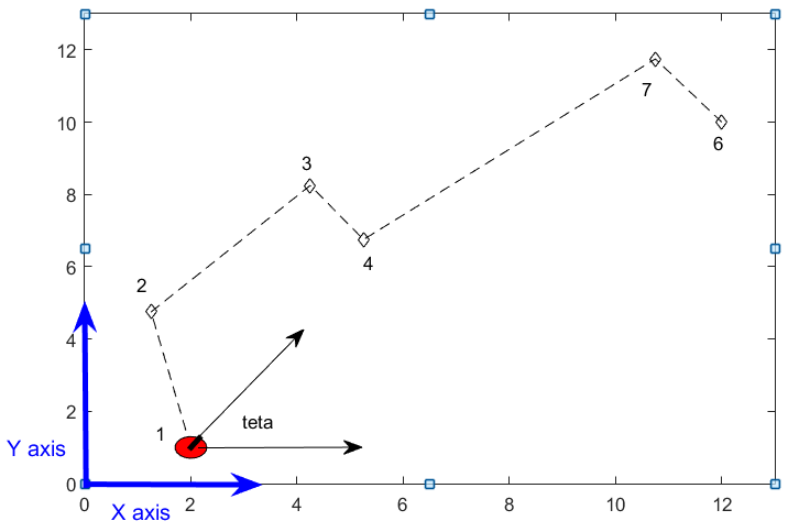


Figure 5.5 : Robot`s position and orientation with respect to the refrence coordinate frame.

for the robot, which it uses it to drive itself along desire path. In MATLAB using PRM codes for a predefined environment with walls as described in section 5.1, a list of possible road maps and finally a short and efficient road map is found for the robot to reach its destination (Figure 5.5).

There are two factors in path tracking methods that has an important impact on the path following results. These two factors are the linear velocity of the robot and the look-ahead distance.

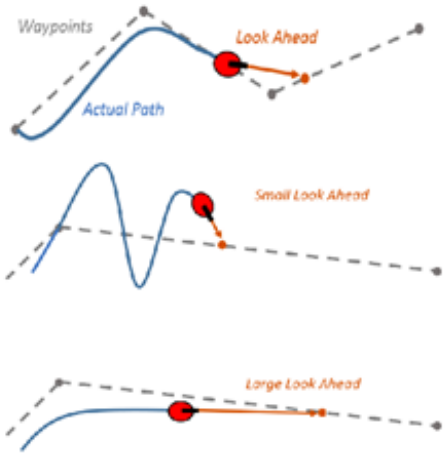


Figure 5.6 : Look ahead distance

The look ahead distance of a robot is how far distance that robot should look from its current location to compute the angular velocity commands. In every path tracking method the actual path of robot will not precisely follow the line between waypoints. As it is shown in figure 5.6, the effects of small look-ahead distances and large ones

are different on robots path. There are two major goals: regaining the path and maintaining on that path. The small look-ahead distance will helps us get to the first goal that is regaining the path more quickly, on the other hand, there will be large amount of overshoot and the robot oscillates along the path. Larger look-ahead distances are used to lessen these large oscillations, however, it may result in large curvatures near corners.

5.2.1 Simulations for a constant look-ahead distance

sing pure pursuit method the robot starts tracking this way from initial point with coordinates $(45, 37)$ to the final position $(15,4)$. Driving robot with this control commands using pure pursuit method will follow the path accurately in lower speeds as shown in figure 5.7. Here the linear velocity speed is considered $0.5\ m/s$. Some sharp corner that follows the pure pursuit algorithm is shown in figure 5.8 more precisely.

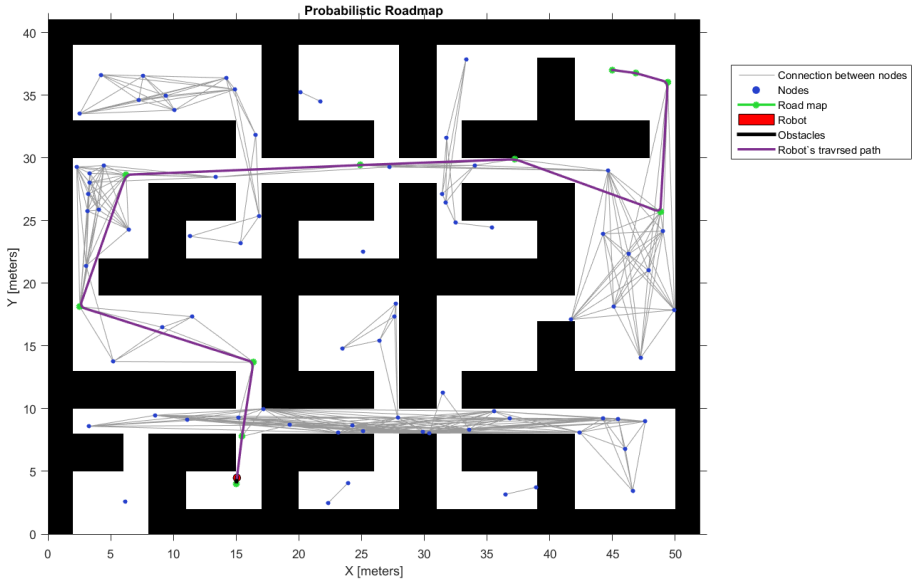


Figure 5.7 : Pure pursuit path tracking method with 0.3 m/s linear velocity

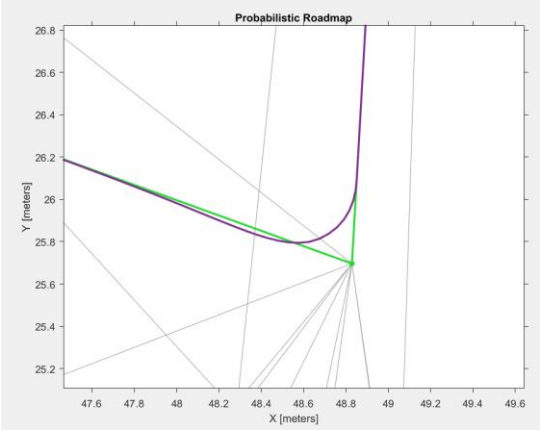


Figure 5.8 : An accurate view of sharp corner turns for the pure pursuit method in 0.5 m/s linear velocity.

As the velocity increases in pure pursuit method, the deviation of the robot from the actual path will be more and the curvatures of the robot to get to the path again will be larger (Figure 5.9).

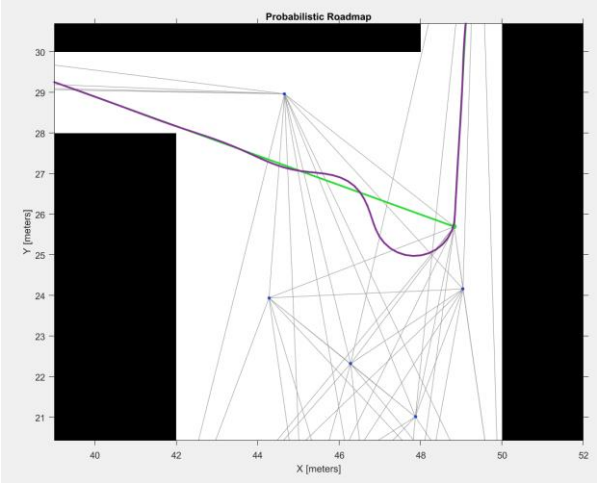


Figure 5.9 : An accurate view of sharp cornet turns for the pure pursuit method in 2 m/s linear velocity

In this section taking a constant value for a look ahead distance, that is 0.5 meters, and changing the linear velocity of the robot to $0.5\ m/s$, $1\ m/s$, $1.5\ m/s$ and $2\ m/s$, we can see the impact of different linear velocities on robots behavior in pure pursuit path tracking method. Firstly, the pure pursuit method then the vector pursuit method will be implemented to these tests. Let us select the first five way points of the road map and survey the results to keep it simpler to continue. What is achieved in the following sections can also be implemented to the all road map from the start point to the end point. Using MATLAB codes and considering four different velocities for the robot,

the robot computes different curvatures in each way point to regain to the path (Figure 5.10). It can be seen as the velocity increases the time for the robot to settle on the path

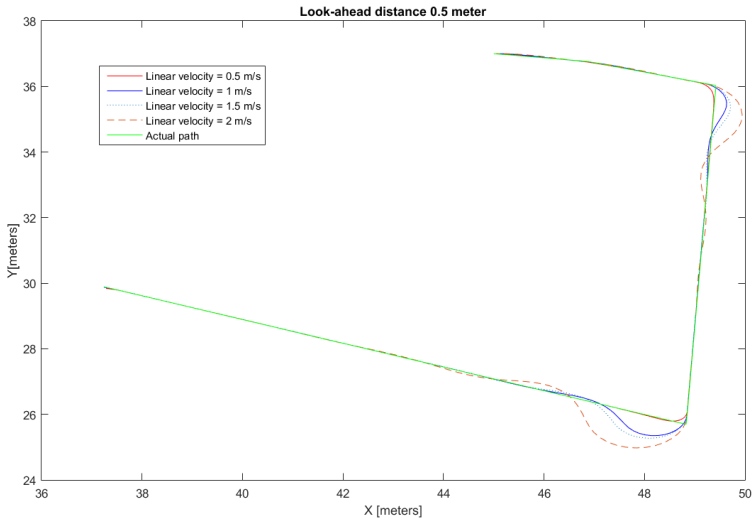


Figure 5.10 : Paths tracked sing pure pursuit method for four different velocities

is also increasing. The red line indicates less oscillations and deviation from the road map for a velocity of 0.5 m/s and the dotted red line has the most oscilatins and error from the actual path for the velocity of 2 m/s. Mapping each point of the actual path (k_{ax}, k_{ay}) to each point of the path surveyed for four different paths gained from four different velocities (k_{px}, k_{py}), we can calculate the error between the main path and each path using equation 5.1 as it is seen in figure 5.11.

$$Error = \Delta = \sqrt{\left(k_{ax} - k_{px}\right)^2 + \left(k_{ay} - k_{py}\right)^2} \tag{5.1}$$

This error Δ, becomes larger for the high velocities and less for lower velocities as it is expected and demonstrated in figure 5.11.

Implementing Vector pursuit to this scenario will result in more accurate path trackings and less errors. The Vector pursuit method also uses the orientation of the point ahead in path tracking and is geometrically meaningful. The path followed using this method for four velocities of the robot 0.5 m/s,1 m/s, 1.5 m/s and 2 m/s, is displayed in figure 5.12 and the error between the actual path and paths surveyed using this method for four velocities that mentioned, is also depicted in figure 5.13. As it is expected in pure pursuit method, the difference of errors between trajectories is much more less than the pure pursuit method because the path is followed more accurate due to pure pursuit algorithms and screw theory.

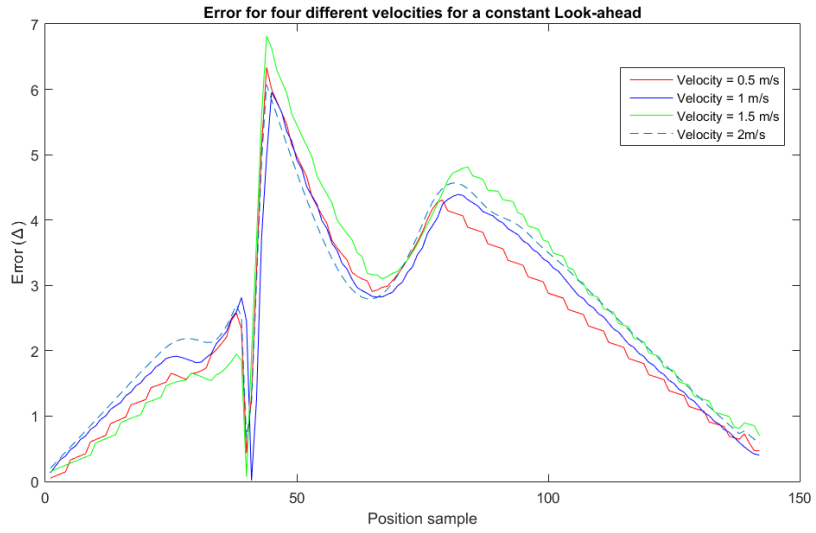


Figure 5.11 : Error between actual path and surveyed paths for four different velocities using Pure pursuit method for a constant look-ahead distance= 0.5 meter

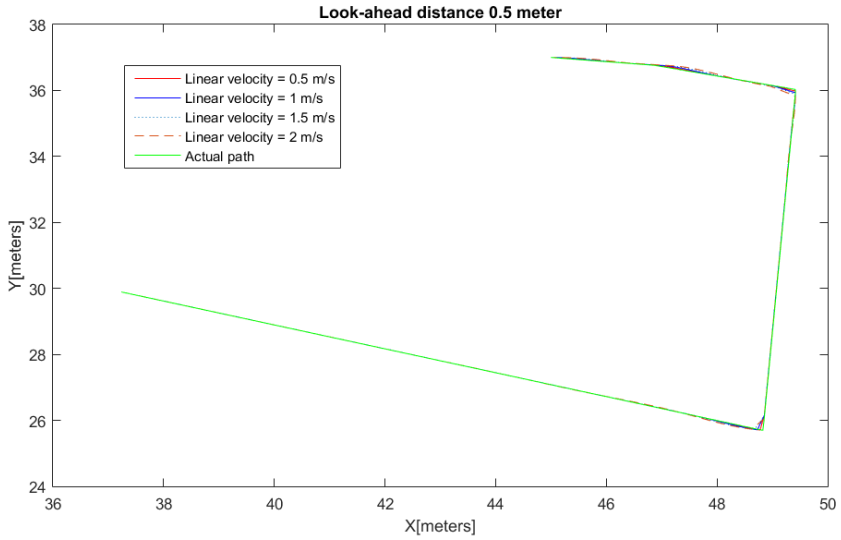


Figure 5.12 : Paths tracked using pure pursuit method for four different velocities

It is important to note that the look-ahead distance,D, is a free choice and it must be chosen in a manner that optimize the performance of the autonomous differential robot. As a result for a constant look-ahead distance in a path planned by RPM method, as the velocity increases, for the pure pursuit method, it takes longer time for the robot to get stable on the path and the standard deviation error increases and it causes remarkable errors in higher velocities. On the other hand, for the vector pursuit method by increasing the velocity in each test it has a smaller overshoot with respect to pure pursuit and the amount of standard deviation error remains approximately constant for

all velocities and it changes slightly for different velocities. (Figure 5.14). Therefore, the vector pursuit method is better that pure pursuit method to handle orientation problems and enduring higher speed errors and take less time to settle down on the road in each corner or turn on the road.

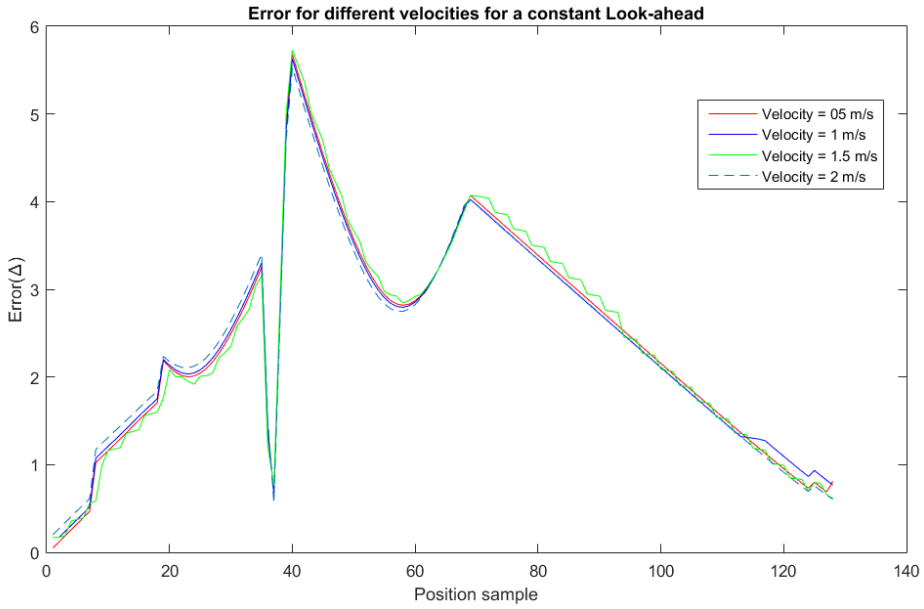


Figure 5.13 : Error between actual path and surveyed paths for four different velocities using Vector pursuit method for a constant look-ahead distance= 0.5 meter

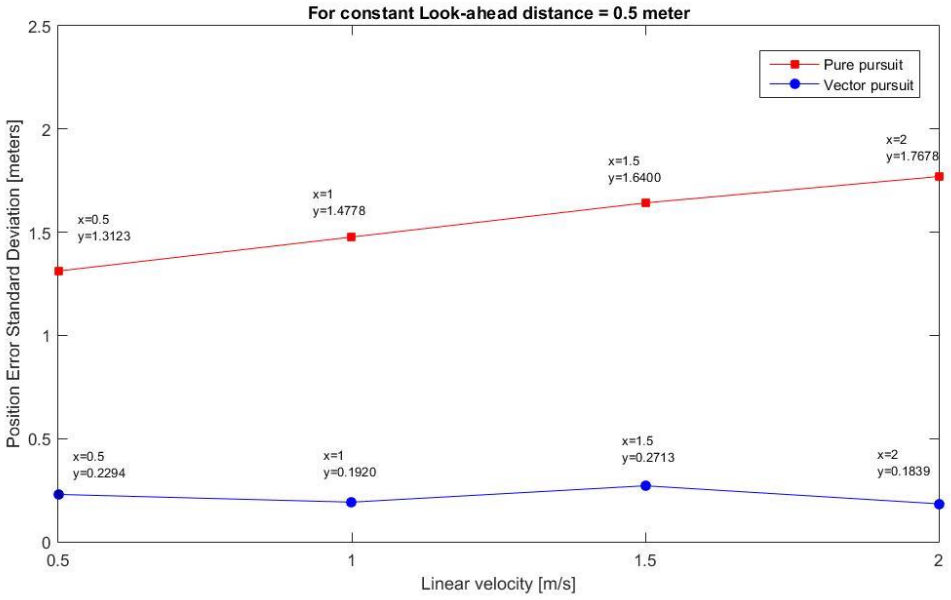


Figure 5.14 : Position error standard deviation for a constant look-ahead= 0.5 m distance comparing pure pursuit method and vector pursuit method

5.2.2 Simulations for a constant velocity

In path tracking methods, the impact of look ahead distance is unneglectable. In addition to being concerned with robot`s linear velocity, the look-ahead distance, D, is also an important concern in path trackings. As mentioned in previous sections a small look-ahead distance may cause large overshoots and instabilities in the path on the other hand larger look ahead distances may follow the path smoother but it will take more time to settle down on the path. In this section, the impact of 4 different look ahead distances, 0.5, 1, 1.3 and 1.7 meters for a constant velocity of 1.5 m/swill be examined. Firstly, the pure pursuit method then the vector pursuit method will be implemented to these tests.

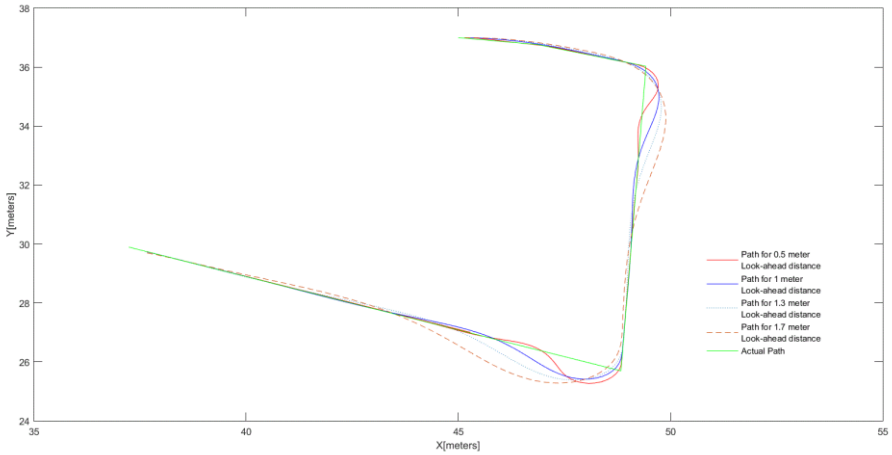


Figure 5.15 : Paths tracked using pure pursuit method for four different look-ahed distances and a constant velocity 1.5 m/s

Again, we select the first five waypoints on the road, and then we can implement the procedure to all of the roadmap. Using MATLAB codes and four different look-ahead distances the robot will compute different curvatures in pure pursuit method to reach to the next look ahead point that (Figure 5.15). As shown in figure 5.15 increasing although increasing the look-ahead distance make a smoother path with less overshoot to settle down on the path, but the it takes more time for the robot to find the path and get to it. On the other hand, small look ahead distances cause more over shoot and less settling time for the robot. These four look ahead distances causes error all over the path with respect to the actual path (Figure 5.16).

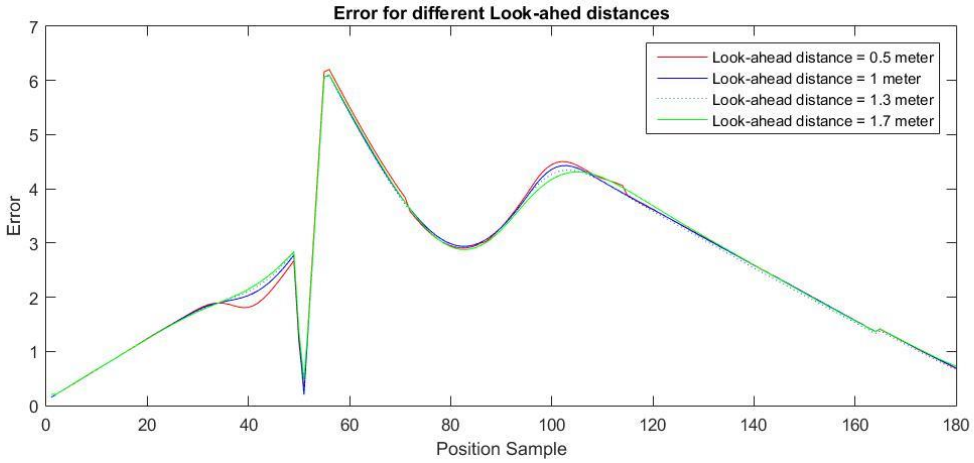


Figure 5.16 : Error between actual path and surveyed paths for four different look-ahed distances using pure pursuit method for a constant velocity $= 1.5\ m/s$

Implementing vector pursuit to this scenario will result in more accurate path trackings, fewer errors and less settling time. The path followed using vector pursuit method for 4 different look-ahead distances, 0.5,1,1,3 and 1.7 meters is depicted in figure 5.17 and the error between actual path and surveyed paths for four given look-ahead distances is shown in figure 5.18. Good results are obtained for vector pursuit method with considering the orientation in each look ahead point for a moving robot on the given path.

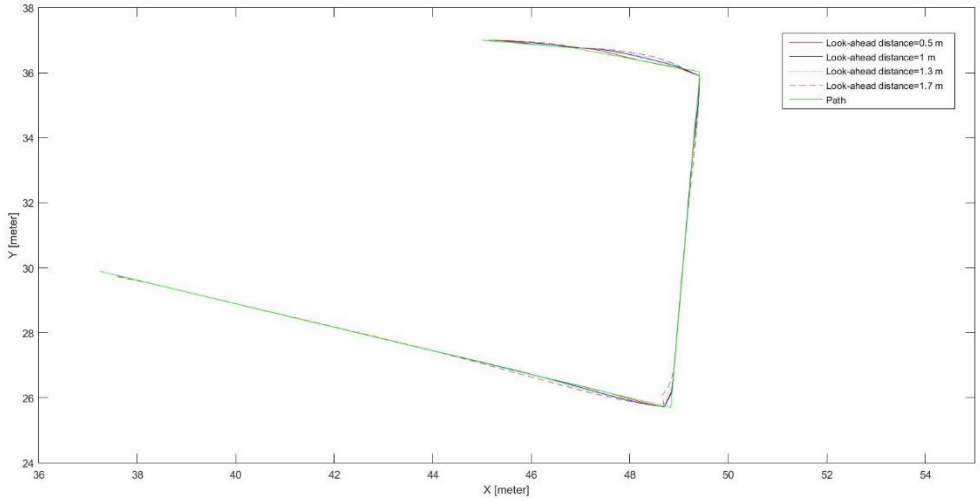


Figure 5.17 : Paths tracked using vector pursuit method for four different look-ahed distances

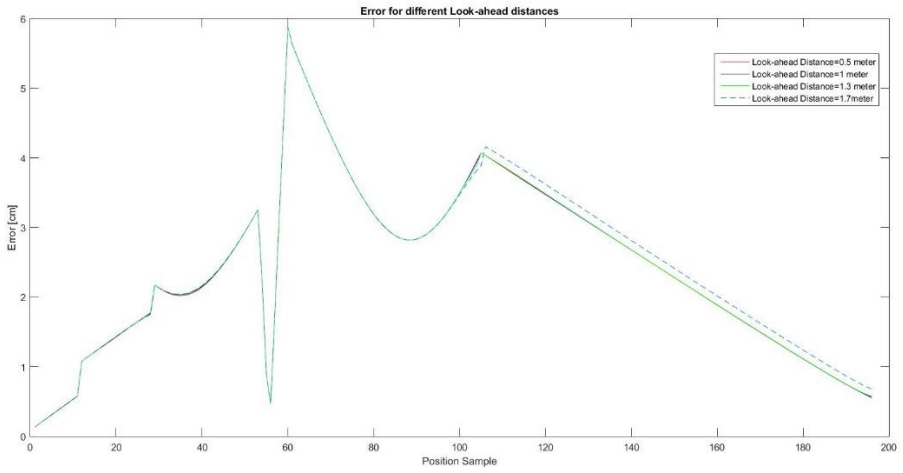


Figure 5.18 : Error between actual path and surveyed paths for four different look-ahed distances using vector pursuit method for a constant velocity$= 1.5\ m/s$

The errors for several D`s as so much little that the graphs are overlapping almost in most of the path. It means that by increasing the look-ahed distance in vector pursuit method there will not be a significant different in robot`s manner.

Comparing position error standard deviation of pure pursuit method and vector pursuit method for a constant velocity$= 1.5\ m/s$ in shown in figure 5.19. From this figure it can be noted for pure pursuit a minimum look-ahed distance to achieve stability for a given velocity can be determined, while vector pursuit is stable over the entire range of look-ahead distances. Note that remaining stale over a range of look-ahed distance is defenatly desirbale and is important. For example suppose pure pursuit path tracking is utlized with a velocity of $1.5\ m/s$ and a look-ahed distance of 1 meter. Good results are obtained under normal conditions. But, if the velocity increases e.g. while goind down a hill, stablity will be a concern for the robot.

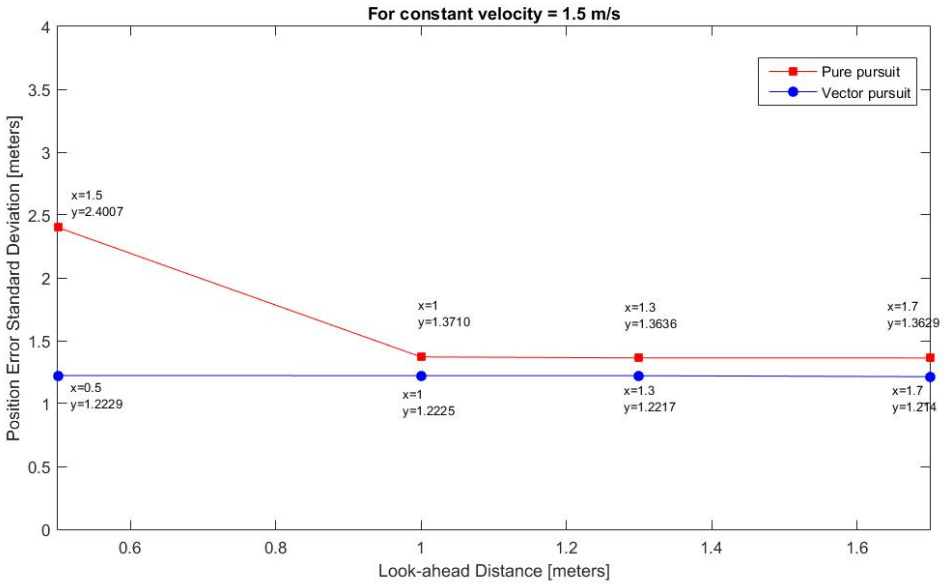


Figure 5.19 : Position error standard deviation for a constant velocity= $1.5 m/s$ comparing pure pursuit method and vector pursuit method

It can be seen from figure 5.19 the pure pursuit method is more flexible to orientations over the road and results in more stable tracking path because the amount of deviation error changes very slightly over the road. Therefore, vector pursuit method is preferd to pure pursuit method.

6. CONCLUSIONS AND RECOMMENDATIONS

A number of path tracking method is discussed in this thesis namely; follow the carrot, pure pursuit and vector pursuit. Considering a previously planned path for a robot using a path planning technique, such as A*, D*, Visibility graph ,cell decomposition and probabilistic road map(PRM) methods helps a robot to find the most optimized and short path during the procedure of getting from its initial point to the destination point. Using PRM path planning technique in MATLAB environment and comparing path tracking methods and simulating their results gives us an understanding of which method is more accurate and efficient for a differential indoor robot.

Two different scenarios are examined in this thesis. In the first scenario the effect of different velocities of robot for a constant look-ahead distance have been studied for two different path tracking methods, pure purist and vector pursuit. By increasing the velocity of the robot to 0.5, 1, 1.5 and 2 m/s for the pure pursuit method it takes longer for the robot to settle down to the road and results in larger oscillations. The position error standard deviation gets larger as the linear velocity increases. On the other hand, for the pure pursuit method in this scenario we approach much robust results and a constant deviation error all over the road that is desirable.

In the second scenario, the effect increasing look-ahead distance has been studied for a given path and two path tracking approaches. Comparing these methods shows that the vector pursuit method based on screw theory generates desired vehicle turning radius based on the robot`s current position and orientation relative to the position and orientation of a point ahead on the planned path using PRM method. The pure pursuit method considers only the position of a point ahead on the path and therefore it is less accurate to exactly follow the path.

Vector pursuit technique uses screw theory to generate vehicle`s desired turning radius based on the robot`s current position and orientation with respect to the position and orientation of the point look-ahead distance ahead of it. To conclude, vector pursuit method is much robust with respect to the sensitivity of velocity to the robot`s current look-ahead distance and also robust with respect to look-ahead distance to the vehicle`s

current speed and is able to handle sudden and large position errors. Vector pursuit method also results in less overshoot and oscillations of the robot all over the path. Therefore, it results for the robot to a less time consuming procedure to reach from the initial position to the goal position.

Another area of possible work could be done using large robots that are bigger than the size of each cell. This problem result in colliding the robot with obstacles, walls or corridors. Using these two path tracking methods will maybe find a good method to avoid this collisions. Another future work can be done by optimization of the look ahead distance. In small look ahead distance oscillation results in path tracking and in a large look ahead distance the robot will not follow the trajectory accurately. Also using PRM method may not find an optimized path from initial point to the end point. All these efforts and studies can be done using Rapidly exploring random tree (RRT) to optimize and find the shortest path for the robot so that there will be less usage of energy sources and a large amount of time will be saved in high time taking procedures.

References

[1] **M. Barbehenn,** "A note on the complexity of Dijkstra`s algorithm for Graphs with weighted Vertices," IEEE Tran. on Computers,, vol. 47, 1998.

[2] **N. J. N. a. B. R. P. E. Hart,** "A formal basis for the heuristic determination of minimum ocst paths in graphs," IEEE Transactionw of Systems Science and Cybernetics , vol. 4, no. 2, pp. 100-107, 1968.

[3] **J. K. G. a. K. S. Nagla,** "A New Approach of Path Planning for Mobile Robots," International Conference on Advances in computing, Communications and Informatics(ICACCI), pp. 863-867, 2014.

[4] **Y.-H. C. a. C.-C. W. Chia-Jun Yu,** "Path Planning Method Design for Mobile Robots," SICE Annual Conference, pp. 1681-1686, 2011.

[5] "Adaptations of the A* algorithm for the computation of fastest paths in deterministic discrete-time dynamic networks," IEEE Trans. on Intelligent Transportation Systems, vol. 3, no. 1, pp. 60-74, 2002.

[6] **A. Stentz,** "Optimal and Efficient Path Planning for Prtially-known Environments," IEEE International Conference on Robotics and Automation, May 1994.

[7] **S. Koenig and M. Likhachev,** "D* Lite," Eighteenth national conference on Artificial intelligence, pp. 476-483, 2002.

[8] **M. v. K. Mark de Berg,** in Computational Geometry Algorithms and Applications, Springer, pp. 323-333.

[9] **N. T.-G. Nora H. Sleumer**, Exact Cell Decomposition of Arrangements Used for Path Planning in Robotics, September 29,1999.

[10] J.-C. Latmbe, ROBOT MOTION PLANNING, SPRINGER SCIENCE+BUSINESS MEDIA, 1991.

[11] **S. U. MAYANK SANGHANI,** “Trajectory Following Robot,” India, 2013.

[12] **O. A. a. C. Thrope,** "Integrated Mobile Robot Control," vol. 1388, September 16, 1990.

[13] **R. e. a. Wallace,** "First Results in Robot Road-Following," CMU-RI-TR-8&4.

[14] **S. S. a. W. S. Dong Hun Shin,** "A partitioned Control Scheme for Mobile Robot Path Tracking".

[15] **J. Morales, J. L. Martinez, M. L. Martinez and A. Mandow,** “Pure-Pursuit Reactive Path Tracking for Nonholonomic Mobile Robots with a 2D Laser Scanner,” EURASIP Journal on Advances in Signal Processing, p. 10, 30 January 2009.

[16] **J. Wit, C. D. Crane III and D. Armstrong,** “Autonomous Ground Vehicle Path Tracking,” Journal of Robotic Systems 21(8), pp. 439-449, 2004.

[17] **Y.-H. C. a. C.-C. W. Chia-Jun Yu,** "Path Planning Method Design for Mobile Robots," SICE Annual Conference, pp. 1681-1686, 2011.

[18] **M. L. Sven Koenig,** "D* Lite".

[19] **A. Stentz,** "The Focussed D* Algorithm for Real-Time Replanning".

[20] **A. Stentz,** "Optimal and Efficient Path Planning for Partially-Known Environments," 1994.

[21] **U. Culha,** "Visibility Graphs," 2016.

[22] **M. v. K. M. O. a. O. S. Mark de Berg**, Computational Geometry Algorithms and Applications, Springer.

[23] **M. Lundgren,** "Path tracking for a Miniature Robot," Sweden.

[24] **O. Amidi,** "Integrated mobile robot control," May 1990.

[25] **B. Y. J. L. a. C. H. DongHyung Kim,** "Analysis of Geometric Path Tracking Method for a Nonholonomic Mobile Robots Based on Vector Pursuit," ICROS-SICE International Joint Conference, pp. 2926-2931, 2009.