

STAGE II REPORT ON

Energy Optimal Path Planning for Mobile Robots

TOWARDS PARTIAL FULFILLMENT OF B.TECH, M.TECH DUAL DEGREE PROGRAMME IN
ENERGY SCIENCE AND ENGINEERING DEPARTMENT

SUBMITTED BY

MANAS CHAUDHARI
09D26003



UNDER THE GUIDANCE OF

Professor Leena Vachhani
Systems and Control Engineering

AND

Professor Rangan Banerjee
Energy Science and Engineering

July, 2014

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources.

I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission.

I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.



(Signature)

Manas Chaudhari

(Name of the student)

09D26003

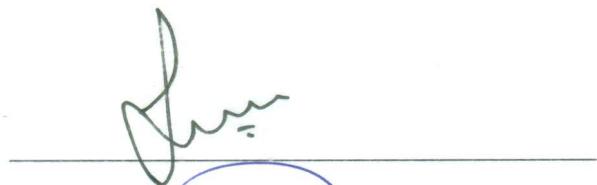
(Roll No.)

Date: 17-7-2014

Approval Sheet

This dissertation entitled "**Energy Optimal Path Planning for Mobile Robots**" submitted by Mr. Manas Chaudhari is approved for the degree of B.Tech.-M.Tech. (Dual Degree) in Energy Science and Engineering.

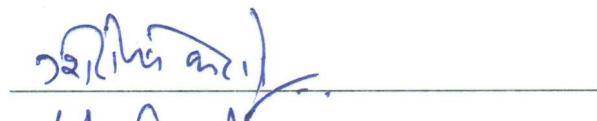
Supervisor:



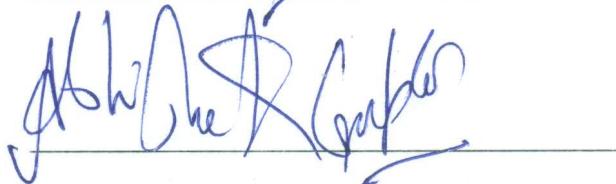
Co-Supervisor:



Internal Examiner:



External Examiner:



Chairman:



Date: 15 July 2014

Place: MUMBAI

Abstract

Mobile robots carry a limited amount of energy. As a result the run time of a robot is limited. In order to improve the run time, energy conservation is essential in mobile robot navigation. The objective of path planning is to determine the optimal trajectory connecting a source point to the destination point which satisfies the constraints of the environment. The goal of this project is to formulate a strategy to reach a destination point by consuming minimum possible energy.

Most methods perform path planning in two stages: waypoints planning and trajectory planning. Waypoints planning involves finding a path consisting of discrete points from source to destination while avoiding obstacles. Trajectory planning involves determining a continuous smooth path between each consecutive pair of waypoints. Because energy consumption depends on the linear and angular velocity profiles of the generated trajectory, the algorithms are computationally expensive. As velocity profiles are unknown during waypoints planning, it is difficult to optimize energy consumption during waypoints planning. This dissertation addresses the following questions: Question 1: Can the expensive algorithms be simplified in order to reduce the computational complexity while still retaining the energy consumption optimality? Question 2: Can an energy consumption criteria which considers the turns in the path be used at the waypoints planning stage to generate waypoints which will lead to minimum energy consumption? Question 3: Which aspect among waypoints planning and trajectory planning has more impact on energy consumption?

A new method for energy efficient trajectory planning using Dubins path having linear time complexity with respect to the number of waypoints has been proposed. Although it does guarantee positive energy savings, the average energy savings are 14.8% compared to an existing trajectory planner which uses Bézier curves. Conventional waypoints planning algorithms such as A* fail to generate optimum results when used with cost functions involving turns. An Edge based search approach has been proposed which considers edges as the nodes during search. Edge based A* and Edge based Theta* methods which are the enhanced versions of A* and Theta* algorithms have been proposed. The proposed Edge based A* and Edge based Theta* methods provide average energy savings of 20.8% and 29.7% compared to their conventional counterparts. The comparison of different methods proposed has shown that waypoints planning has a greater impact on energy savings compared to trajectory planning.

Contents

List of Figures	iii	
List of Tables	iv	
1	Introduction	1
1.1	Motivation	1
1.2	Problem Formulation and Assumptions	2
2	Literature Survey	3
2.1	Research Questions	5
2.2	Methodology	6
3	Energy Optimal Trajectory Planning	7
3.1	Trajectory Planning using Bézier Curves	7
3.1.1	Robot Model	7
3.1.2	Energy Model	7
3.1.3	Determining Minimum Energy Path	9
3.1.4	Determining Minimum Energy Trajectory	10
3.1.5	Challenges	11
3.1.6	Possible Improvements	13
3.2	Trajectory Planning using Dubins path	13
3.2.1	Types of paths	13
3.2.2	Computing Dubins path	14
3.2.3	Minimizing Energy Consumption	14
3.2.4	Constant Linear Velocity Condition	14
3.2.5	Energy Components	14
3.2.6	Energy Calculation	15
3.2.7	Determining optimum trajectory	16
3.3	Results	17
3.3.1	Simulation	17
3.3.2	Results	17
3.3.3	Energy Calculation	17
3.3.4	Comparison	20
3.4	Conclusion	21
4	Waypoints Planning	23
4.1	The conventional A* search approach	23
4.1.1	An Example Problem	24
4.1.2	Problems with turn based cost functions	25
4.2	The proposed Edge based A*	26

4.2.1	Proof	27
4.2.2	Time Complexity	28
4.2.3	Comparison with A*	28
4.3	The Theta* approach	29
4.3.1	The proposed Edge based Theta*	32
4.3.2	Further enhancement on the proposed Edge based Theta*	32
4.4	Improvement of energy saving using Edge based approaches	34
4.4.1	Comparison of search algorithms	38
5	Conclusions	42
5.1	Impact of trajectory planner	42
5.2	Impact of waypoints planner	43
5.3	Conclusions	43
A	Calculating Dubins path	45
A.1	Common Calculations	46
A.2	Determining Path Length & Critical Values	46
A.2.1	RSR	46
A.2.2	LSL	49
A.2.3	RSL	50
A.2.4	LSR	51
A.2.5	RLR	53
A.2.6	LRL	53
B	Admissible Heuristics for A* search	55

List of Figures

2.1	Results from Wang et al. [1]	3
2.2	Optimal and suboptimal velocity profiles with respect to energy consumption [2]	4
2.3	Path generation based on (a) minimal energy, (b) minimum distance, (c) larger distance from obstacles [3]	5
2.4	Dubins path	6
3.1	Two wheeled differential drive robot [3]	8
3.2	Dubins path examples	13
3.3	GUI for path planning	17
3.4	Trajectories	18
3.5	Comparison of implementation results with published results	19
3.6	Histogram showing frequency of energy savings obtained	20
3.7	Best case scenario trajectory	21
3.8	Worst case scenario trajectory	22
4.1	An example search problem	24
4.2	Example	26
4.3	Start and End Nodes	27
4.4	Diagram for Proof	28
4.5	Comparison of paths generated by Normal A* search (left) and Edge based A* search (right). Origin is on the top left corner in each grid.	30
4.6	Paths generated by Normal Theta* and Edge based Theta*	32
4.7	Paths generated by Edge based Theta* and Enhanced Theta*	36
4.8	Maximum energy savings for Edge based Theta*	39
4.9	Histograms showing frequency of energy savings for different methods and average savings compared with A*	40
4.10	Minimum energy savings for Edge based Theta* (All paths are identical)	41
5.1	Probability distribution of percentage energy savings using A* with Dubins path compared to A* with Bézier curves (Average savings: 14.8%)	42
5.2	Energy Savings comparison using different combinations of proposed methods	44
A.1	An RSR trajectory	45
A.2	RSR trajectory for different radii	47
A.3	RSR trajectory for high r_0	48
A.4	LSL trajectory	49
A.5	RSL trajectory	50
A.6	LSR trajectory	51
A.7	RLR trajectory	52
A.8	LRL trajectory	54

List of Tables

1.1	Specifications of robots [4, 5]	2
3.1	Parameter values	18
3.2	Computation Times	20
3.3	Best case energy comparison	21
3.4	Worst case energy comparison	22
4.1	Solution to search problem from Figure 4.1 using A* search	25
4.2	Normal A* calculations for Example 1 (Figure 4.5a)	31
4.3	Edge based A* calculations for Example 1 (Figure 4.5a)	31
4.4	Computation comparison of Normal A* and Edge based A* methods	31
4.5	Calculations for Theta* algorithm for example in Figure 4.6a	34
4.6	Calculations for Theta* algorithm for example in Figure 4.6a	35
4.7	Calculations for Enhanced Theta* algorithm for example in Figure 4.7b	37
4.8	Energy profile comparison for best case scenario in Figure 4.8 (All energy values are in Joules and savings are calculated with respect to A*)	39
4.9	Energy profile comparison for worst case scenario in Figure 4.10 (All energy values are in Joules and savings are calculated with respect to A*)	39

Chapter 1

Introduction

Robot navigation is the robot's ability to reach desired target locations in its environment. Navigation is essential for any task which has to be performed at a specific location in the environment. Mobile robots have several applications such as remote exploration of unreachable areas, fetching objects etc. The navigation itself can either be manual, for example human controlled or it can be autonomous in which the robot reaches the target location on its own. Autonomous navigation consists of planning motion of the robot on a path. The path may or may not be known a priori. When the path is unknown, the autonomous navigation needs to solve the problems of path planning as well as motion planning. The goal of the project is to develop a path planning strategy to reach the navigation goal with minimum energy consumption.

Path Planning

The objective of path planning is to determine the trajectory which the robot follows to reach the goal. The path planning techniques depend on whether the environment is known or not. In case of unknown environment, the robot detects obstacles by processing sensor data. Common techniques which are used for path planning include potential functions, A* search, Dijkstra's algorithm, Belman-Ford algorithm. In the potential functions method, a navigation function is designed such that there is a global minima at the goal. The goal is reached by following the negative gradient of the function. The other techniques are search based techniques which are used for finding the least cost path in a weighted graph.

1.1 Motivation

Mobile robots require a portable source of energy. Batteries are very commonly used. As the energy from a battery is limited, the run time of the robot in one battery charge is limited to a few hours. Table 1.1 shows a survey of robots and their respective run times. Using a battery with higher capacity also adds to the weight of the robot which in turn results in higher power consumption. As a result, in order to increase the run time of robots, it is essential to reduce the power consumption of the robot.

A major part of the total energy consumption in a mobile robot occurs at the motors for navigating the robot. Most methods in literature target at minimizing distance or time during path planning. However, the shortest path need not be the most energy efficient path. For example, traversing a short path with many turns could consume more energy than a straight longer path. Also, the motors have different efficiency at different speeds. Considering these during motion planning will help in reducing the energy consumption for the navigation task. This project focuses on the task of reducing energy consumption in navigation for a given robot.

Robot Name	Weight kg	Payload kg	Voltage V	Battery Capacity Ah	Uptime h
Amigo Bot	3.6	1	3.3/5/12	2.1	2 - 3
Fire Bird V	1.3	-	9.6	2.1	2
Fire Bird XI	15	10	11.1	6 × 4	4 - 5
Pioneer 3-DX	9	17	12	7.2 × 3	8 - 10
Neobotix MP-500	70	50	24	38	10

Table 1.1: Specifications of robots [4, 5]

Apart from improving the robot's run time, there are several advantages of modeling the energy consumption of a robot. For example, given a navigation target, if the estimated energy requirement for performing the task exceeds the available energy, the robot can navigate to the docking station, charge its battery till the required level and then complete the pending task.

1.2 Problem Formulation and Assumptions

The project considers path planning for a differential wheeled robot in a known environment. The frictional coefficient of the ground at each point is known. The objective is to compute a path for reachable target location using minimum energy or declare that the target is unreachable. In particular, the robot must detect the unreachable target condition in finite time. It is assumed that the robot does not gain energy by braking and energy gets dissipated in the form of heat when the wheel speed decreases.

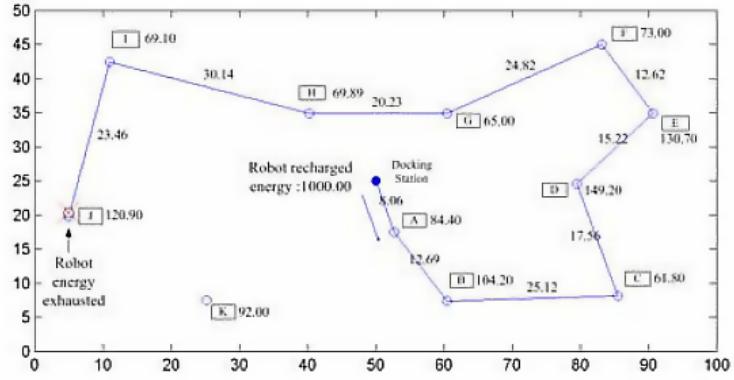
The next chapter reviews different methodologies from literature which target similar problems. It explains the gaps in existing methods and gives an overview of the methods proposed in this report.

Chapter 2

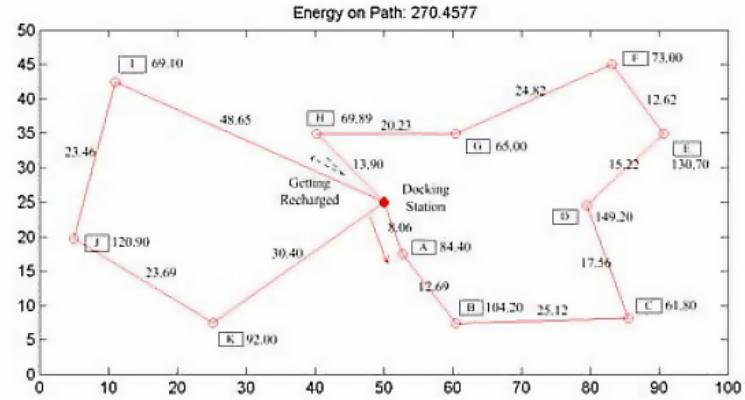
Literature Survey

This chapter describes existing techniques from literature in the domain of energy optimal planning for robots.

Wang et al. [1] have considered a problem in which robots are powered by batteries which can be charged at a docking station. The staying-alive and energy-optimal path planning has been addressed by determining the optimum order of way-points aiming toward minimizing the energy consumption. Figures 2.1a and 2.1b show the result of the work.



(a) Path without staying alive consideration [1]



(b) Path with staying alive consideration [1]

Figure 2.1: Results from Wang et al. [1]

Sun and Reif [6] have worked on the problem of determining energy optimal path on a terrain which minimizes energy losses due to friction also considering overturn dangers at different slopes by defining impermissible travel directions at each point on the terrain.

Ooi and Schindelhauer [7] have worked on energy optimal path planning for a single robot. This robot establishes wireless communication with a radio base station. An approximate algorithm has been developed to find energy optimal path minimizing the energy consumption for mobility and the energy consumption for communication which increases with the transmission distance.

Distance traveled by a robot has direct impact on the energy consumption. Mei et al. [8] have proposed a method for energy efficient robot exploration. Energy consumption has been reduced by reducing repeated coverage. Plonski et al. [9] have discussed the problem of energy optimal path planning in case of solar powered robots. Solar maps built using the sensor information are used to maximize the battery life of the robot.

Barili et al. [10] have worked on energy saving motion strategies for an autonomous mobile robot in environments like civil buildings which contain unpredictable obstacles. The strategy aims to minimize the change in velocity. Chitsaz et al. [11] have worked on determining optimum paths with minimum wheel rotation for differential drive vehicles. The optimum path may not be a shortest distance path because shortest distance path does not imply that minimum distance covered by the wheels.

Determination of energy optimum path based on planning requires an energy consumption model. Morales et al. [12] have developed a model for power consumption of skid-steer tracked robots. Power losses due to dynamic friction and the power losses at motor drivers have been considered in the model. Mei et al. [13] have proposed an energy efficient motion planning technique using a relationship of power consumption of motors with their speeds. The energy consumption for acceleration and turns at different velocities is compared for different routes.

Tokekkar et al. [2] have worked on the problem of determining velocity profiles for car-like robots in order to minimize the energy consumption for travelling on a given path. The method considers the energy consumed by the DC motors. The model considers the energy required to overcome internal and load friction, the energy dissipated in the resistive winding and the mechanical power output at the shaft. The energy consumption of a DC motor depends on the angular velocity and the acceleration. An optimization is performed to determine the velocity profile which minimizes the energy consumption. Figure 2.2 shows the comparison of optimal and suboptimal velocity profiles.

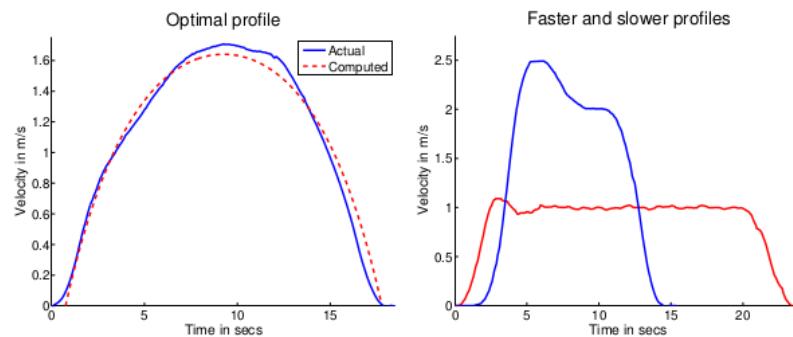


Figure 2.2: Optimal and suboptimal velocity profiles with respect to energy consumption [2]

Energy consumption of a mobile robot depends on the linear and angular velocity profiles of the trajectory. The energy computations involve second derivatives of the trajectory co-ordinates

with respect to time. Liu and Sun [3] have proposed a method for energy optimal path planning which involves a model for formulating energy consumption of a wheeled robot for kinetic energy transformation and overcoming friction. The waypoints are determined using A* search with an energy based cost function involving ground resistance coefficients. The energy model has been used to determine the optimal parameters for the trajectory formed using Bézier curves passing through the waypoints, which minimize energy consumption.

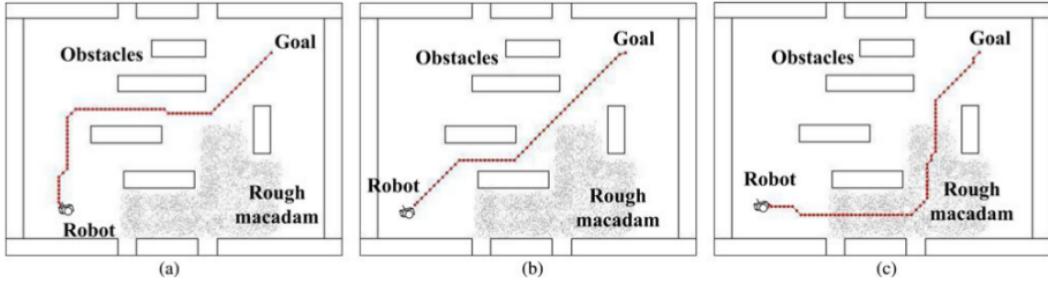


Figure 2.3: Path generation based on (a) minimal energy, (b) minimum distance, (c) larger distance from obstacles [3]

2.1 Research Questions

Unlike path length which only depends on the position of the path, energy consumption is a complex criteria which depends on the position of the path as well as the velocity and acceleration profiles at each point on the path. As a result, the methods for optimizing energy tend to be expensive in terms of computation. As robots need to compute the trajectories repeatedly, use of computationally expensive planning techniques becomes infeasible.

As there are infinitely many possibilities of paths between a source and a destination point, most methods divide path planning into two parts: waypoints planning and trajectory planning. In waypoints planning, the goal is to find a set of points which the robot can go through in order to reach the destination. Obstacle avoidance is handled in waypoints planning. Once the set of waypoints is determined, the trajectory planning is performed for each pair of consecutive waypoints to generate smooth trajectories.

Because a trajectory provides more information compared to a list of waypoints, more accurate energy consumption estimates can be computed in the trajectory planning stage. As the waypoints are the basis for the trajectory, the choice of waypoints can greatly influence the energy consumption. Different strategies exist in the literature for waypoints planning and trajectory planning, but trajectory planning techniques are more expensive because of their wider search space.

The waypoint selection method proposed by Liu and Sun [3] does not consider the number of turns or any factor related to the shape of the expected trajectory while the number of possible trajectories considered increases exponentially with the number of waypoints. The trajectory computation is far more expensive compared to the waypoints planning.

This gives rise to a few questions:

- *Can the expensive algorithms be simplified in order to reduce the computational complexity while still retaining the energy consumption optimality? How much energy is lost for using a simpler method? If simpler algorithm causes higher consumption, do higher computations provide savings enough to balance their energy cost?*

- Most commonly used method for searching waypoints is the A^* search. The cost functions for A^* search are usually dependent on length of the path. Can this method be extended to search using an energy consumption cost function?
- Which of the two aspects are more important from the perspective of energy saving? Better waypoints or better trajectories? Will a computationally expensive waypoints planner with a simple trajectory planner be more efficient than a simple waypoints planner with an expensive trajectory planner?

2.2 Methodology

Dubins [14] solved the problem of finding the shortest curve that connects two points in the two-dimensional Euclidean plane (i.e. x-y plane) with a constraint on the curvature of the path and with prescribed initial and terminal tangents to the path. It is proved that the shortest path is formed by joining curves with minimum possible radius and straight lines. Figure 2.4 shows a simple example. Because the Dubins path can be constructed with lesser parameters, a trajectory planner using Dubins path would be less expensive than a trajectory involving more parameters such as the Bézier curve. Chapter 3 proposes a method for planning energy efficient trajectories using Dubins paths with different curvature constraints. The method proposed by Liu and Sun [3] has been used as a baseline for comparing the energy consumption.

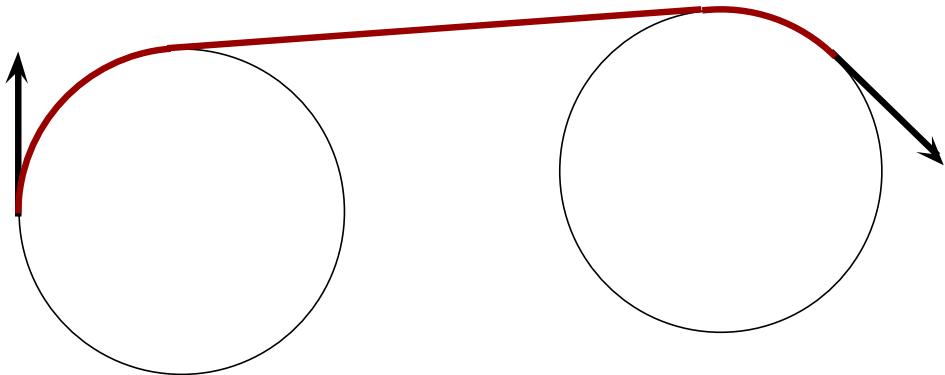


Figure 2.4: Dubins path

Chapter 4 focuses on techniques for waypoints planning. The issues with A^* search from the perspective of energy efficient path planning have been identified and a new Edge based method has been proposed which is capable of using cost functions involving turns. The paths generated by A^* search [15] are constrained on the edges of the grid. This limitation causes the paths to have more number of turns. Nash et al. [16] have proposed the Theta* algorithm for computing any-angle paths, which are shorter and have lesser number turns than the paths generated by A^* . An Edge based Theta* method for generating energy efficient paths has also been proposed.

Chapter 5 summarizes the results of the proposed waypoint and trajectory planners and performs an analysis to determine which aspect of the path has more impact on the energy consumption.

Chapter 3

Energy Optimal Trajectory Planning

This chapter explains the energy optimal path planning method proposed by Liu and Sun [3]. The method uses a model for computing the energy consumed by the motors of the robot. Path planning is performed in two steps. First, a path is calculated using an A* search variant which uses energy based constraints. Next, the smooth trajectory is determined along the path by selecting appropriate velocity and arrival time at the waypoints such that energy consumption is minimized. A new method for trajectory planning using Dubins Path has been proposed. The results obtained by the proposed method have been compared with the Bézier curves method proposed by Liu and Sun [3].

3.1 Trajectory Planning using Bézier Curves

3.1.1 Robot Model

In a differential wheeled robot, the motion is achieved using two separately driven wheels as shown in Figure 3.1. The direction of the robot is changed by selecting different speeds for the wheels. The wheel velocities v_{left}, v_{right} are related to the linear and angular velocities of the robot v, ω by (3.1).

$$\begin{bmatrix} v_{left} \\ v_{right} \end{bmatrix} = \begin{bmatrix} 1 & -b \\ 1 & b \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}, \quad (3.1)$$

where u and v are the linear and angular velocities respectively, and b is the distance between the wheel and the midpoint of the axis connecting the two wheels. The next section describes the energy model of the system under consideration.

3.1.2 Energy Model

The energy consumption of the motors consists of the energy required to change the kinetic energy of the robot and the energy required to overcome the friction. Apart from the motors, there is also a constant power consumption at the sensors, controllers and the central computing device. If the total power consumption of all sources apart from motors is P_{other} , the energy consumption is calculated as

$$E_{other} = P_{other} \times \text{Time}. \quad (3.2)$$

Kinetic Energy

This is the energy required to accelerate or decelerate the robot. The total kinetic energy consists of linear and rotational components. If $v(t)$ and $\omega(t)$ are the linear and angular velocities

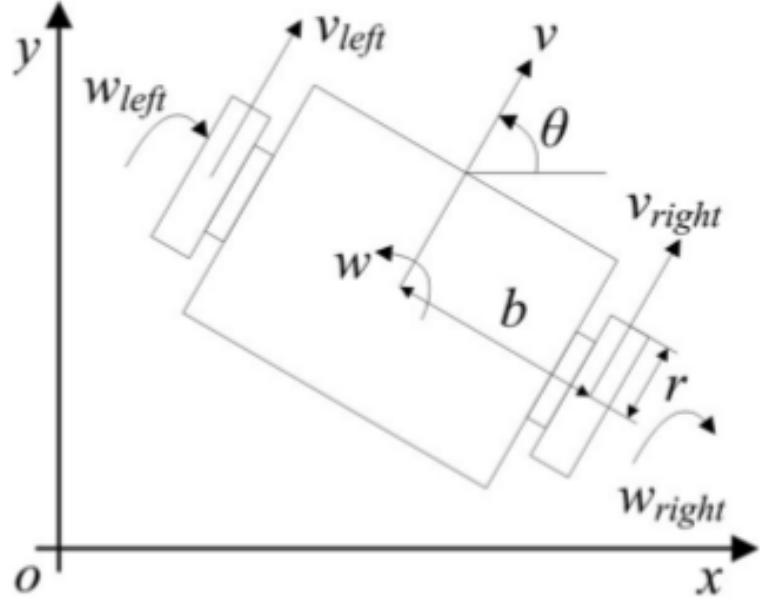


Figure 3.1: Two wheeled differential drive robot [3]

respectively expressed as functions of time t , kinetic energy is given by

$$\begin{aligned}
 E_{kinetic} &= \frac{1}{2}mv(t)^2 + \frac{1}{2}I\omega(t)^2 \\
 &= \int_t \left(d\left(\frac{1}{2}mv(t)^2\right) + d\left(\frac{1}{2}I\omega(t)^2\right) \right) \\
 &= \int_t (mv(t)a(t) + I\omega(t)\alpha(t)) dt,
 \end{aligned}$$

where $a(t), \alpha(t)$ are the linear and angular accelerations respectively expressed as functions of time and m, I are the mass and moment of inertia respectively. Generally, the transformation of kinetic energy to electric energy is not efficient. Hence, it is assumed that energy is dissipated in the form of heat during deceleration when $v(t)a(t) < 0$ or $\omega(t)\alpha(t) < 0$. Thus, the energy consumed by the robot is expressed as

$$E_{kinetic} = \int_t (m \max\{v(t)a(t), 0\} + I \max\{\omega(t)\alpha(t), 0\}) dt. \quad (3.3)$$

Energy to overcome friction

The energy required to overcome traction resistance which is due to the rolling friction of the two wheels is also provided by the motors. The power consumption of the two wheels is modeled as

$$\begin{bmatrix} P_{left} \\ P_{right} \end{bmatrix} = \mu mg \begin{bmatrix} |v_{left}| \\ |v_{right}| \end{bmatrix},$$

where μ is the friction co-efficient and g is the acceleration due to gravity.

From (3.1), we have

$$\begin{bmatrix} P_{left} \\ P_{right} \end{bmatrix} = \mu mg \begin{bmatrix} |v(t) + b\omega(t)| \\ |v(t) - b\omega(t)| \end{bmatrix}.$$

Total energy consumption due to traction resistance is given by

$$\begin{aligned} E_{res} &= \int_t \mu mg(|v(t) + b\omega(t)| + |v(t) - b\omega(t)|) dt \\ &= 2\mu mg \int_t \max\{|v(t)|, |b\omega(t)|\} dt. \end{aligned}$$

Therefore, the energy consumption of the motor is

$$E_{motor} = \int_t (m \max\{v(t)a(t), 0\} + I \max\{\omega(t)\alpha(t), 0\} + 2\mu mg \max\{|v(t)|, |b\omega(t)|\}) dt. \quad (3.4)$$

The next section describes the calculation of the energy optimal path using this energy model.

3.1.3 Determining Minimum Energy Path

The energy optimal path is calculated by introducing an energy based constraint in the cost function of A* search algorithm [15]. The environment is represented as a grid of cells. A cell can be either free or occupied with obstacles. Each cell is a node for the A* search. The search starts with the starting node. The order in which the nodes are searched depends on the cost function, which for node k is given by

$$f(k) = g(k) + h(k),$$

where $g(k)$ is the cost of the path from starting node to node k and $h(k)$ is the heuristic estimate of the cost of path from node k to goal node. As the nodes are explored in the increasing order of $f(k)$, when the goal is reached, it is known that the path to goal has the minimum possible value of $f(k)$. In this method, $f(k)$ is chosen to be proportional to energy consumption along the path. Hence, minimizing $f(k)$ minimizes energy consumption.

The cost $g(k)$ is computed as the sum of the cost to reach previous node $g(k-1)$ and the cost to reach from $g(k-1)$ to $g(k)$. Because the robot velocity is not known during path planning, the kinetic energy component is not included in the cost.

$$g(k) = g(k-1) + 2\mu_{k-1,k} mg s_{k-1,k},$$

where $\mu_{k-1,k}$ is the friction co-efficient and $s_{k-1,k}$ is the distance between nodes $k-1$ and k . In order to avoid paths which are too close to obstacles, a penalty factor $\rho(k) \in [0, 1]$ is introduced to the cost function given by

$$g(k) = g(k-1) + 2\mu_{k-1,k} mg \frac{s_{k-1,k}}{\rho(k)}, \quad (3.5)$$

where $\rho(k)$ is defined as

$$\rho(k) = \begin{cases} 1 & d_{obs} > D_{safe}, \\ \frac{d_{obs}-b}{D_{safe}-b} & b < d_{obs} \leq D_{safe}, \\ 0 & d_{obs} \leq b, \end{cases}$$

where d_{obs} is the distance to the nearest obstacle, D_{safe} is the safe distance beyond which no penalty is required and b is the minimum clearance required for the vehicle.

The next section explains the computation of an energy optimal smooth trajectory based on the path obtained in the A* search.

3.1.4 Determining Minimum Energy Trajectory

The trajectory is computed as a series of connected Bézier curves. Bézier curve is a smooth cubic parametric curve. A series of waypoints is chosen from the path obtained from the A* search. The parameters for the curves are determined from the positions and orientations of the waypoints such that energy consumption along the complete trajectory is minimized.

Waypoint Selection

Waypoints are selected in an iterative process. Initially, the start and the goal and the knee points on the generated path are selected as waypoints. If any two neighbouring waypoints are sufficiently close, they are combined while a new waypoint is inserted if the path segment is extremely long.

Determining Curve Parameters

The position of a waypoint W_i is denoted by $q_i = [X_i, Y_i, \theta_i]^T$. The trajectory based on Bézier curve between waypoints W_{i-1} and W_i is represented as

$$\begin{aligned} x(u) &= (1-u)^3 X_{i-1} + 3u(1-u)^2 X_{ai} + 3u^2(1-u) X_{bi} + u^3 X_i \\ y(u) &= (1-u)^3 Y_{i-1} + 3u(1-u)^2 Y_{ai} + 3u^2(1-u) Y_{bi} + u^3 Y_i, \end{aligned}$$

where $(x(u), y(u))$ denote a point on the trajectory expressed as a function of $u \in [0, 1]$ which is an intermediate variable proportional to time and $X_{ai}, X_{bi}, Y_{ai}, Y_{bi}$ are the parameters to be determined. As u changes from 0 to 1, $(x(u), y(u))$ changes continuously from W_{i-1} to W_i . If T_i denotes the time of arrival at waypoint W_i , u is expressed as a function of time t by $u_i = \frac{t-T_{i-1}}{T_i-T_{i-1}}$.

$$\begin{aligned} t = T_{i-1} &\Leftrightarrow u_i = 0 \Leftrightarrow (x(u_i), y(u_i)) = (X_{i-1}, Y_{i-1}) \\ t = T_i &\Leftrightarrow u_i = 1 \Leftrightarrow (x(u_i), y(u_i)) = (X_i, Y_i). \end{aligned}$$

Considering orientation constraints at the waypoints, we have

$$\begin{aligned} \frac{dx(u_i)}{dt} \bigg|_{t=T_{i-1}} &= \frac{3(X_{ai} - X_{i-1})}{T_i - T_{i-1}} = V_{i-1} \cos \theta_{i-1} \\ \frac{dy(u_i)}{dt} \bigg|_{t=T_{i-1}} &= \frac{3(Y_{ai} - Y_{i-1})}{T_i - T_{i-1}} = V_{i-1} \sin \theta_{i-1} \\ \frac{dx(u_i)}{dt} \bigg|_{t=T_i} &= \frac{3(X_i - X_{bi})}{T_i - T_{i-1}} = V_i \cos \theta_i \\ \frac{dy(u_i)}{dt} \bigg|_{t=T_i} &= \frac{3(Y_i - Y_{bi})}{T_i - T_{i-1}} = V_i \sin \theta_i, \end{aligned} \tag{3.6}$$

where V_i denotes the robot velocity at waypoint W_i . Equation (3.6) is rearranged to obtain the relations

$$\begin{aligned} X_{ai} &= X_{i-1} + \frac{1}{3}(T_i - T_{i-1})V_{i-1} \cos \theta_{i-1} \\ Y_{ai} &= Y_{i-1} + \frac{1}{3}(T_i - T_{i-1})V_{i-1} \sin \theta_{i-1} \\ X_{bi} &= X_i - \frac{1}{3}(T_i - T_{i-1})V_i \cos \theta_i \\ Y_{bi} &= Y_i - \frac{1}{3}(T_i - T_{i-1})V_i \sin \theta_i. \end{aligned} \tag{3.7}$$

Thus, knowing velocities and arrival times at the waypoints, the parameters for the Bézier curves are calculated.

Computing Energy Consumption

Based on the energy model (3.4), calculating energy consumption requires knowing linear & angular velocities as well as linear & angular accelerations at each point of time in the trajectory. From $x(u_i), y(u_i)$, the linear velocity is derived as

$$v(u_i) = \left(\left(\frac{dx(u_i)}{dt} \Big|_t \right)^2 + \left(\frac{dy(u_i)}{dt} \Big|_t \right)^2 \right)^{\frac{1}{2}}. \quad (3.8)$$

The curvature of the trajectory $\delta(u_i)$ is derived as

$$\delta(u_i) = \frac{\frac{d^2y(u_i)}{du_i^2} \frac{dx(u_i)}{du_i} - \frac{d^2x(u_i)}{du_i^2} \frac{dy(u_i)}{du_i}}{\left(\left(\frac{dx(u_i)}{du_i} \right)^2 + \left(\frac{dy(u_i)}{du_i} \right)^2 \right)^{\frac{3}{2}}}. \quad (3.9)$$

Angular velocity $\omega(u_i)$ is calculated as the product of curvature and linear velocities.

$$\omega(u_i) = \delta(u_i)v(u_i). \quad (3.10)$$

The linear and angular accelerations can be calculated by differentiating linear and angular velocities. Thus, we have $v(u_i), \omega(u_i), a(u_i), \alpha(u_i)$ expressed in terms of $V_i, T_i, V_{i-1}, T_{i-1}$. Now, the motor energy consumption to reach from waypoint W_{i-1} to W_i is calculated as

$$E_{i-1,i,motor} = \int_{t=T_{i-1}}^{t=T_i} (m \max\{v(u_i)a(u_i), 0\} + I \max\{\omega(u_i)\alpha(u_i), 0\} + 2\mu mg \max\{|v(u_i)|, |b\omega(u_i)|\}) dt. \quad (3.11)$$

Minimizing Energy Consumption

Let \mathbb{V} and \mathbb{T} by the sets of V_i and T_i respectively for all waypoints. The total energy consumption depends on \mathbb{V} and \mathbb{T} . The values for \mathbb{V} and \mathbb{T} that result in minimum energy consumption are determined by solving the unconstrained optimization problem

$$\underset{\mathbb{V}, \mathbb{T}}{\text{minimize}} \left(P_{\text{other}} T_N + \sum_{i=0}^{N-1} E_{i-1,i} (V_{i-1}, T_{i-1}, V_i, T_i) \right).$$

The next section describes the limitations and the challenges involved in the implementation of this method.

3.1.5 Challenges

The approach consists of an energy model, an energy based cost function for A* search and an optimization problem for determining the parameters of the trajectory composed of several Bézier curves. The challenges faced in implementing this method and its limitations are discussed in this section.

Computationally Expensive

In trajectory planning, the search space increases exponentially with waypoints. If we denote the number of waypoints by n_w , the size of linear velocity search space by n_v and the size of

time intervals search space by n_t , the number of possible trajectories T_n is

$$\begin{aligned} T_n &= n_v^{n_w} n_t^{n_w-1} \\ &= \frac{1}{n_t} (n_v n_t)^{n_w}. \end{aligned}$$

If we consider 3 different velocities, 3 different time intervals and 10 waypoints, then number of possible trajectories is approximately 1.1×10^9 .

Inappropriate choice of decision variables

The decision variables for the optimization problem are the velocity V_i and arrival time T_i at all waypoints. The method has framed the optimization problem such that V_i and T_i are independent of each other. Arrival times as decision variables can take an extremely wide range of values. It is very difficult to form the search space for arrival times that would include the optimal configuration.

Energy Model Accuracy

The model considers that when acceleration is in the opposite direction of velocity i.e. when kinetic energy decreases, energy is lost in the form of heat. This assumption should be applied to the individual wheels instead of applying to the entire vehicle. Consider the case when the robot is moving on a circular trajectory such that $v > b\omega, \omega > 0$. Therefore, the wheel velocities are given by

$$\begin{aligned} v_{left,0} &= v - b\omega \\ v_{right,0} &= v + b\omega. \end{aligned}$$

At the end of the circular trajectory, the robot follows a straight trajectory. After the transition, the linear velocity is v while the angular velocity becomes zero. Wheel velocities at this stage would be equal to v . As per the model proposed by Liu, the energy consumption to perform this transition should be zero because the angular acceleration is opposite to the angular velocity ($\frac{d\omega}{dt} \omega < 0$). However, the left wheel speed increases by an amount of $b\omega$ and hence, it would require some amount of energy. This energy need not be converted into kinetic energy. It could be dissipated directly in the form of heat as overall, the kinetic energy of the robot has decreased. This situation does not occur during linear acceleration as both wheel velocities increase or decrease simultaneously.

Sequential Optimization

During path planning by A* search, the energy based cost function does not consider complete energy. It considers only the component required for overcoming the resistance. Because of this, the overall motion planning is biased towards reducing this energy component. The method first minimizes this component of energy consumption, then minimizes the component required for changing kinetic energy. The sequential optimization does not guarantee optimum results. For example, it is possible that a straight trajectory through a region with higher resistance can be more optimal than a trajectory with many turns through a region with less resistance.

These limitations motivate the need for developing better energy optimal path planning techniques. The next section discusses some of the possible improvements which can overcome these limitations.

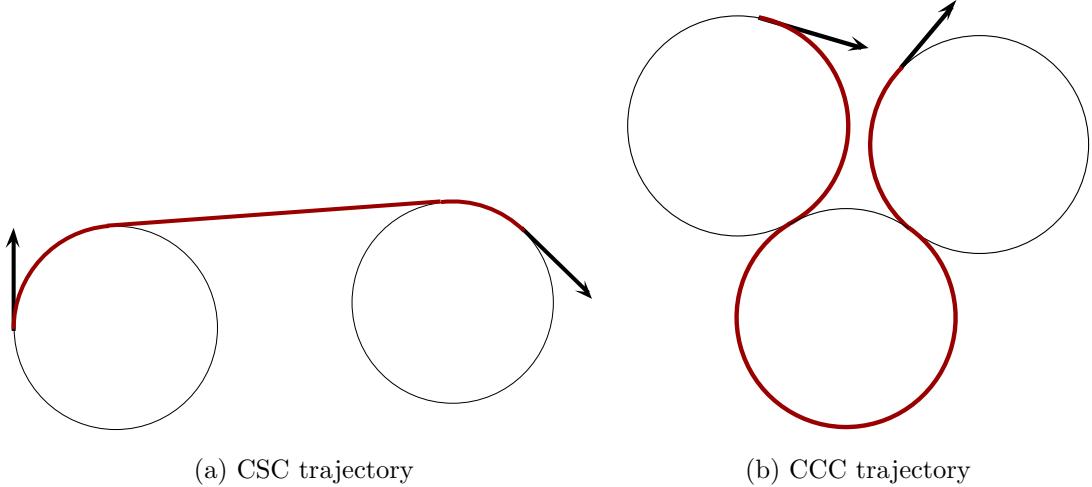


Figure 3.2: Dubins path examples

3.1.6 Possible Improvements

Even though the choice of V_i does not impose any constraints on T_i , choosing a value V_i does give some rough idea about the optimal T_i because the end points of the trajectory segment are fixed and known. For example, if it is known that two waypoints W_{i-1} and W_i are 1m apart, then for $V_{i-1} = V_i = 0.5\text{m/s}$, choosing time interval of 10s would mean that the vehicle would have to slowdown and then accelerate back to 0.5m/s. Thus, the initial kinetic energy is lost due to deceleration and more energy is required to accelerate. As a result, in most cases, this choice of time interval would give inefficient results. This information can be used to reduce the optimization search space and also to improve the accuracy of the solution by using variable arrival time search space depending on the chosen velocity. Instead of applying the energy balance on the entire robot, it can be applied on individual wheels. This model will be more accurate as the energy balance would also consider the kinematics of the robot.

The method discussed in this chapter computed the trajectory between two waypoints as a Bézier curve. The next section describes a new method in which the trajectory is computed as a Dubins path [14]. A Dubins path ¹ consists of three parts, two of which are circular arcs while the third part can be a straight line or a circular arc. Because the kinetic energy of the robot changes at only finite number of points in case of Dubins path, it has several advantages compared to Bézier curve.

3.2 Trajectory Planning using Dubins path

In 1957, Dubins [14] proved that the shortest trajectory with an upper bound on curvature and with fixed orientation at endpoints consists of circular arcs of maximum radius and straight lines. Figure 3.2 shows examples of Dubins path. An additional constraint is that the trajectory cannot reverse its direction at any point or the vehicle can move only in the forward direction.

3.2.1 Types of paths

There are two categories of paths: CCC (Curve-Curve-Curve) and CSC (Curve-Straight-Curve) as shown in Fig. 3.2, depending on whether the path is a combination of 3 curves or 2 curves

¹The path in Dubins path is actually a type of trajectory and should not be confused with the type of path which is generated by the A* search

and a straight line respectively. Further each curve corresponds to either a left turn L or a right turn R . Thus, we have six possible types of trajectories.

$$T \in \{ LRL, RLR, LSL, LSR, RSL, RSR \} \quad (3.12)$$

The first two types in the above equation belong to the CCC category while the last four belong to the CSC category.

3.2.2 Computing Dubins path

Determining the path consists of determining the optimal choice between the six combinations in (3.12). The process of computing Dubins path is explained in Appendix A.

3.2.3 Minimizing Energy Consumption

This section proposes a method for determining the energy optimal trajectory through a given sets of waypoints. The proposed method uses an approach similar to the one discussed in Section 3.1. An energy model is formulated for computing the energy consumption of the trajectory. An optimization problem is framed with trajectory parameters as the decision variables and energy consumption as the cost function.

3.2.4 Constant Linear Velocity Condition

If the linear velocity of the robot is increased, its kinetic energy increases, resulting in higher energy consumption. If the linear velocity of the robot is decreased, the total time for reaching the goal increases. As energy consumption by other devices is proportional to time, the total energy consumption increases. Thus, changing the velocity leads to a higher energy consumption. Hence, the proposed method considers trajectories with a constant linear velocity.

Now, the energy consumption of the robot depends on the angular acceleration. Considering the Dubins path, each segment of the trajectory is either a circular arc or a straight line. Thus, the angular velocity can have only three possible values.

$$\omega(t) = \begin{cases} -\omega_0 & \text{on left circle} \\ 0 & \text{on straight line,} \\ \omega_0 & \text{on right circle} \end{cases} \quad (3.13)$$

where $\omega_0 = v/r_0$, where v is the linear velocity and r_0 is the minimum turning radius.

3.2.5 Energy Components

This method considers energy consumption by the motors and other devices similar to the model discussed in section 3.1.2. The power consumption by other devices is considered as constant. The motor energy consumption consists of two components. One component is the energy required for changing the kinetic energy of the robot and other is the energy required for overcoming the ground resistance. The first component depends on the linear and angular velocity profiles of the planned trajectory while the second component depends on the length of the trajectory.

The kinetic energy of the robot at any point of time is given by

$$KE(t) = \frac{1}{2}m v(t)^2 + \frac{1}{2}I \omega(t)^2, \quad (3.14)$$

where m and I are the mass and moment of inertia respectively of the robot. Now, the energy consumption would be equal to the increase in kinetic energy. When kinetic energy decreases, it is assumed that it is completely dissipated as heat and energy consumption is zero.

The energy required to overcome ground resistance is calculated as

$$P_{res}(t) = \mu mg L_0, \quad (3.15)$$

where L_0 is the path length.

3.2.6 Energy Calculation

As the linear velocity is considered constant (Section 3.2.4), a change in kinetic energy occurs only when the angular velocity changes. But, from (3.13), the angular velocity can have only three values. Hence, the energy consumption is non-zero for following transitions:

$$\{ 0 \rightarrow +\omega_0, 0 \rightarrow -\omega_0, \pm\omega_0 \rightarrow \mp\omega_0 \}.$$

For the first two cases, it is clear that the energy consumption will be equal to $\frac{1}{2}I\omega_0^2$. For the third case, the transition can be broken into two parts: $\pm\omega_0 \rightarrow 0 \rightarrow \mp\omega_0$. In the first part, as the kinetic energy is decreasing, the energy consumption is zero. In the second part, the energy consumption is equal to the change in energy which is equal to $\frac{1}{2}I\omega_0^2$.

The energy consumption also depends on the angular velocity transition at the way-point. The initial angular velocity at the way-point is equal to the angular velocity based on the trajectory planned between this and previous way-points. Thus, if the previous trajectory is RSL , the initial angular velocity is $-\omega_0$. If the current trajectory is LSL , then the change in kinetic energy at the starting point is $\frac{1}{2}I\omega_0^2$ which is added to the energy consumption. On the other hand, if the initial angular velocity is ω_0 , then the kinetic energy does not change at the starting point. While planning trajectory between way-points W_{i-1} and W_i , the initial angular velocity i.e. angular velocity at W_{i-1} would be equal to the endpoint angular velocity of the trajectory between previous pair of way-points W_{i-2}, W_{i-1} . Thus the transition from a XXL to RXX trajectory or from XXR to LXX requires energy $\frac{1}{2}I\omega_0^2$ where X denotes any type of segment.

Thus, the kinetic energy component of motor energy consumption is expressed as

$$E_k = E_{initial} + E_{trajectory}$$

$$= \begin{cases} \frac{1}{2}I\omega_0^2 & XXL \rightarrow RXX \\ \frac{1}{2}I\omega_0^2 & XXR \rightarrow LXX \\ 0 & \text{otherwise} \end{cases} + \begin{cases} \frac{1}{2}I\omega_0^2 & LSL \\ \frac{1}{2}I\omega_0^2 & LSR \\ \frac{1}{2}I\omega_0^2 & RSL \\ \frac{1}{2}I\omega_0^2 & RSR \\ I\omega_0^2 & LRL \\ I\omega_0^2 & RLR \end{cases} \quad (3.16)$$

The energy consumption for overcoming friction, E_f is calculated from the path length using (3.15). Determining the path length has been explained in Appendix A.

As the linear velocity is constant, the time for reaching the goal is proportional to the length of the path. Thus, energy consumption by other devices is calculated by

$$E_{other} = P_{other} \times L/v, \quad (3.17)$$

where v is the linear velocity, L is the length of the path and P_{other} is the power consumption by other devices.

3.2.7 Determining optimum trajectory

The kinetic energy E_k decreases as turning radii r increases since angular velocity ω is inversely proportional to turning radii r . On the other hand, the friction energy E_f consumption increases as turning radii r increases since there is an increase in the path length when turning radius r is more. As turning radius r approaches to zero, the trajectory approaches towards a straight line between start and goal. An optimum value of turning radius r is determined which results in minimum total energy consumption considering lower and upper bounds for the turning radii. These bounds are given as follows:

Let r_{min} denote the lower bound on the value of turning radius r . The r_{min} is the minimum turning radius of the robot at linear velocity v . Dubins path for $r = r_{min}$ is calculated. Let T_{min} be the time consumption for the calculated trajectory.

The upper bound for the circle radii for a Dubins trajectory type D is determined by solving the critical conditions for existence of D type of Dubins trajectory. For example, for an RSR type of trajectory, the critical condition occurs when the two circles touch each other internally. If radius of first circle is r_i , then the upper bound on the radius of second circle is given by

$$r_{max} = -\frac{-2r_i(x_i - x_{i+1})\sin\theta_i + 2r_i(y_i - y_{i+1})\cos\theta_i + (x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}{2(r_i - (y_i - y_{i+1})\cos\theta_{i+1} + (x_i - x_{i+1})\sin\theta_{i+1} - \cos(\theta_i - \theta_{i+1}))}, \quad (3.18)$$

where $(x_i, y_i, \theta_i), (x_{i+1}, y_{i+1}, \theta_{i+1})$ are the configurations at way-points W_i and W_{i+1} respectively. A negative value of the expression in (3.18) implies that the critical condition never exists. In this case, a global maximum turning radius is used.

The length of the trajectory is determined from the turning radius and initial & final configurations. However, the length would be discontinuous as the trajectory type changes at the critical values of turning radius r . Therefore, the length calculation is broken into parts such that each part is continuous. As the relation between length and radius is different for different types of categories, the length function for a Dubins trajectory type D is described by (3.19).

$$L_i = L_D(q_i, q_{i+1}, r_i) \quad \forall i = 0, \dots, N, \quad (3.19)$$

where q_i denotes the configuration (x_i, y_i, θ_i) at way-point W_i , q_0 and q_N represent source and target configurations respectively and r_i is the turning radius under consideration at way-point W_i . Note that the function L_D would have different domains of turning radius r for different trajectory type D . Also, these domains would be mutually exclusive as one combination of (q_i, q_{i+1}, r_i) corresponds to a unique D . The length functions and critical values for different trajectories along with their derivations are explained in Appendix A.

Thus, for each way-point, the optimal value of turning radius r_i is calculated as

$$r_i = \arg \min(E_k + (2\mu m g + P_{other}/v) L_D(q_i, q_{i+1}, r)), \quad (3.20)$$

where function $\arg \min$ returns the argument of the function. Note that q_i and q_{i+1} represent the known configurations of the way-points and the trajectory type D is determined from (q_i, q_{i+1}, r_{min}) which results in a unique choice of the Dubins trajectory. The equation (3.20) renders the desired radius for the Dubins path between waypoints W_i and W_{i+1} that would result in minimizing the total energy consumption.

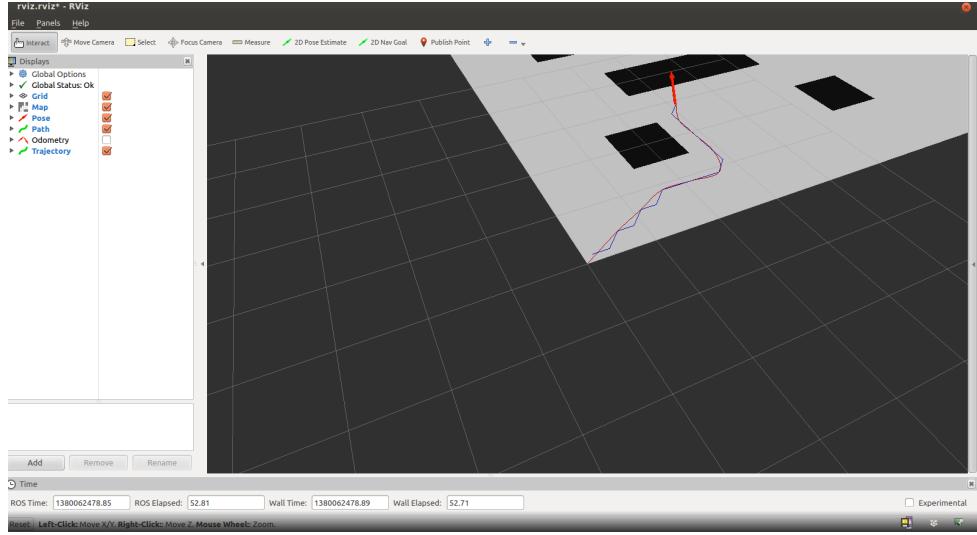


Figure 3.3: GUI for path planning

3.3 Results

3.3.1 Simulation

The proposed method is implemented using Gazebo with ROS in Ubuntu 12.04 64-bit. The experimental scenario is developed that has similarity with the scenario used in Liu and Sun [3]. The method developed in Liu and Sun [3] is also implemented to compare our results. The energy consumption for the resultant trajectories are calculated analytically using the energy model. Fig. 3.4a shows the path obtained using A* search in the simulation environment.

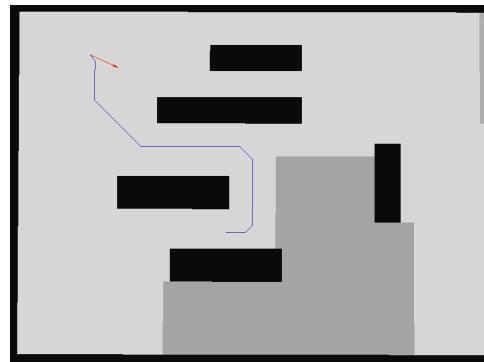
The worst case energy consumption for a given path is calculated as follows: the robot moves on a straight line with a constant linear velocity toward a way-point. On reaching the way-point, it halts, turns to face the next way-point with maximum angular velocity and then continues moving toward the next way-point. Figures 3.4b and 3.4c show the energy optimal trajectories generated using the method suggested in Liu and Sun [3] and proposed method respectively. The simulation also provides the energy consumption profiles of the two trajectories as well as the worst case path. We next compare the energy consumption and computation time of proposed method with the Bézier curves method proposed by Liu and Sun [3].

3.3.2 Results

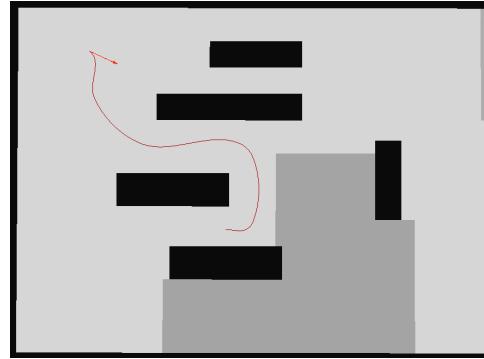
Liu and Sun [3] have calculated third order Bézier curve at a way-point for minimizing energy along the shortest path. In this section we compare our results with the worst case and that obtained by Bézier curve method. We have used the parameters used by Liu and Sun [3] for computing the energy optimal trajectories. Table 3.1 lists the values of different parameters.

3.3.3 Energy Calculation

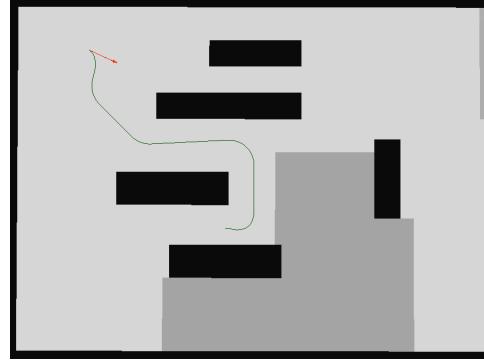
For the Bézier curves, the energy consumption is calculated by differentiating x and y co-ordinates. The velocity is calculated from the difference in x and y co-ordinates. Angular velocities are calculated from the second derivative of x and y co-ordinates. Linear and angular accelerations are determined from the derivatives of linear and angular velocities respectively. Then, the energy consumption is calculated based on the energy model discussed in Section 3.1.2.



(a) Path obtained using A^* search



(b) Smoothened path using Bézier curve suggested by [3]

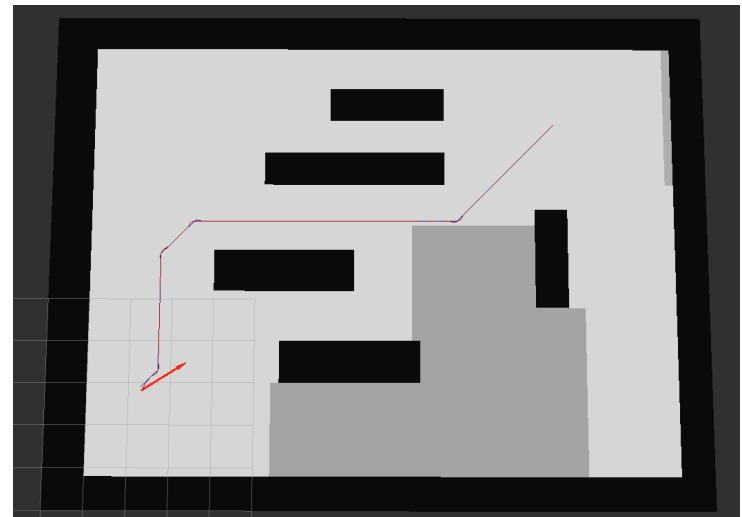


(c) Smoothened path using proposed method

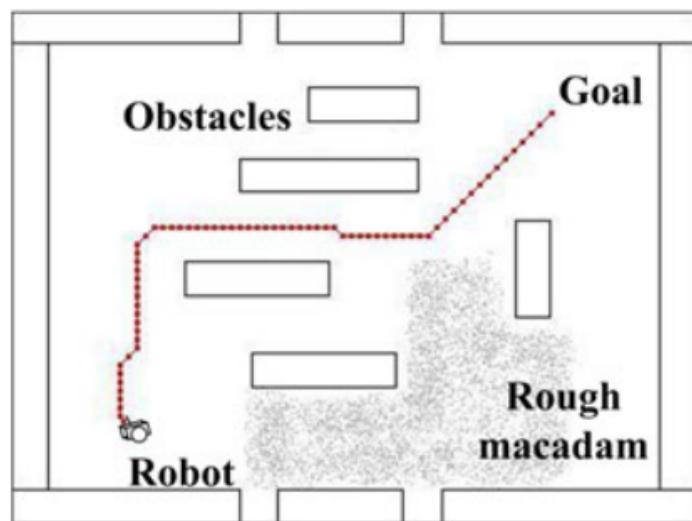
Figure 3.4: Trajectories

Mass	9 kg
Moment of Inertia	0.16245 kg m^2
Robot radius b	0.185 m
μ_{min}	0.051
μ_{max}	0.1078
$P_{constant}$	17.7 W
Minimum turning radius	0.1 m
Map resolution	80×60

Table 3.1: Parameter values



(a) Results from implementation



(b) Published result [3]

Figure 3.5: Comparison of implementation results with published results

For the Dubins path, the energy consumption is calculated from the circle radii and the type of trajectory as described in Section 3.2.6.

3.3.4 Comparison

Computation Time

The same set of waypoints is used for trajectories computed by both the methods. Both the methods involve optimization. The computations are proportional to the search space of optimization. If r_n denotes the size of radius search space and w_n denotes the number of way-points, the proposed method requires energy computation r_n number of times per way-point. As the radius computation is independent at different way-points, the method has a computation complexity of $O(w_n r_n)$. In energy optimal path planning method using Bézier curves, the decision variables are the linear velocities v_i and arrival times T_i at each way-point. Because the parameter optimization is performed globally, choice of v_i, T_i is not independent of v_j, T_j for $i \neq j$. Hence, the computation complexity is $O((v_n t_n)^{w_n})$ where v_n denotes the number of different linear velocities in the search space. The computation times for both methods on a machine with Intel Core i7 (1.60 GHz) processor are tabulated in Table 3.2.

Bézier curves				Dubins path		
w_n	v_n	v_t	Time(s)	w_n	r_n	Time(s)
3	5	5	0.24	8	100	0.04
6	5	5	2.92	75	100	0.18
7	5	5	16.32	118	1000	2.18
8	5	5	86.05			

Table 3.2: Computation Times

The proposed method has very low computation time compared to the Bézier curve method because the proposed method has linear computation complexity whereas the Bézier curve method has exponential computation complexity.

Energy Savings

Multiple simulation results are obtained using different methods between multiple start and goal positions. Figure 3.6 shows the proportion of cases with different energy savings.

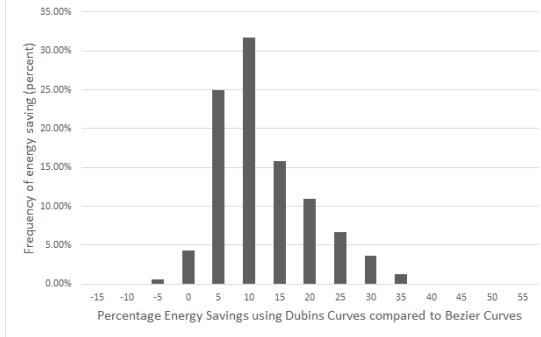


Figure 3.6: Histogram showing frequency of energy savings obtained

Parameters	Bézier curve	Proposed	A^*
E_{total}	245	153	452
$E_{ke,linear}$	12.64	6.48	45.36
$E_{ke,angular}$	151.78	65.68	327.50
E_{res}	31.45	30.19	29.70
E_{other}	49.62	50.43	49.80
v_{max}	1.27	1.20	1.20
ω_{max}	21.50	15.53	24.00
length	3.41	3.42	3.36
time	2.80	2.85	2.81

Table 3.3: Best case energy comparison

In the best case, the energy consumption of the trajectory using Dubins path is 37.8% less than that using Bézier curves as shown in Fig. 3.7. The figure shows the A^* trajectory using blue color, the Bézier curve trajectory using red color and the proposed method's trajectory using green color. Table 3.3 tabulates the energy consumption by different methods. The worst case of energy computation is considered for the path with sharp turns obtained by A^* . The differential drive mobile robot stops for taking sharp turns. The table also tabulates different energy component for different methods. The tabulated results show that the energy saving is mainly because the proposed trajectory has lesser angular velocities at the turns, which is indicated by the lesser kinetic energy E_k component in Table 3.3.

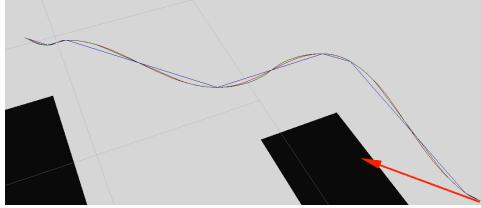


Figure 3.7: Best case scenario trajectory

In the worst case, the energy consumption of the trajectory obtained by the proposed method is 5.73% more than that obtained using the Bézier curve method. This case is shown in Fig. 3.8. The path in blue color shows the way-points. The trajectory in red color represents the energy optimal trajectory through the way-points obtained using Bézier curves. The trajectory in green color represents the energy optimal trajectory through the way-points obtained using Dubins path.

Referring to Table 3.4, the comparison of different energy components shows that the slightly higher energy consumption of proposed method is mainly due to the increase in kinetic energy. The maximum angular velocity of trajectory obtained by the proposed method is higher than the maximum angular velocity of trajectory obtained by the Bézier curve method.

3.4 Conclusion

A method for determining energy optimal trajectory using Dubins curves from a given set of waypoints in linear time complexity has been proposed in this chapter. The method minimizes energy on the basis of an energy model which considers the motor energy consumption for

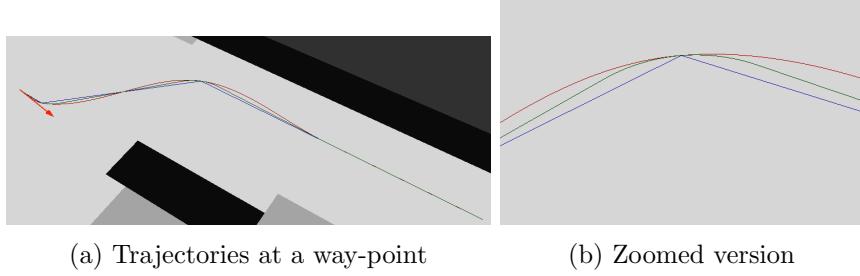


Figure 3.8: Worst case scenario trajectory

Parameters	Bézier curve	Proposed	A^*
E_{total}	140	148	390
$E_{ke,linear}$	9.07	6.48	32.40
$E_{ke,angular}$	6.91	17.46	233.93
E_{res}	46.56	46.32	46.16
E_{other}	77.11	77.39	77.16
v_{max}	1.27	1.20	1.20
ω_{max}	7.07	13.89	24.00
length	5.27	5.25	5.23
time	4.36	4.37	4.36

Table 3.4: Worst case energy comparison

transforming kinetic energy and overcoming ground resistance.

Even though the method does not guarantee energy savings, the energy losses are limited to 5.73% (worst case loss) compared to energy consumption of trajectory using Bézier curves suggested by Liu and Sun [3]. In the best case, the method provides energy savings of up to 37.8% compared to the method by Liu and Sun [3]. The proposed method provides average energy savings of 14.8%. Power consumption for other devices has been considered constant. However, in reality, the power consumption of the computing device depends on the number of computations performed. As a result, reduced computations also reduces energy consumption which has not been accounted in our energy calculations. The actual energy savings are expected to be greater than our conservative estimates.

In this chapter, the friction based cost function was used for selecting the waypoints. This process can be improved by considering turns in the search cost function. The next chapter discusses the issues in implementing this with the regular A^* search and proposes Edge based methods which yield better results.

Chapter 4

Waypoints Planning

The previous part of this report focused on generating energy efficient trajectories from a given set of waypoints. This part focuses on determining waypoints such that the energy required to travel through them is minimized. Common methods use A* search based on a distance based cost function to determine the waypoints. In order to optimize energy, in the previous chapter, A* search with a friction based cost function was used to determine the waypoints. The friction based cost function involved the product of friction co-efficient of the surface and the length of path. However, energy cost highly depends on the amount of turns in the path and turns were not optimized by the friction based cost function. When using turn based cost functions, the conventional search approaches fail to give optimum results. An Edge based search approach which overcomes this limitation has been proposed in this chapter. The Edge based approach is not specific to a particular search algorithm. For analysis, the A* search has been used.

This chapter explains the problems with conventional A* search and how the Edge based A* method solves them. Next, the Edge based approach is applied to the Theta* algorithm. An energy cost function for use with Edge based search algorithms has been proposed. A*, Theta* are examples of graph search algorithms. The next section provides an introduction to the conventional A* method and explains the reason due to which it cannot be used with turn based cost functions.

4.1 The conventional A* search approach

Consider a graph as shown in Figure 4.1. The points are connected to each other with edges. Each edge has an associated cost. A* search algorithm can search for a path from a source point to a destination point with minimum cost where the cost of a path is the sum of the costs of all edges in the path. In addition to the cost for edges, the A* search also requires the heuristic cost for each point. The heuristic is treated as an approximate cost for reaching the goal from that point. A* search requires a heuristic to be admissible in order to produce optimum paths. The conditions for admissible heuristics have been discussed in Appendix B.

Terminology

- x_s, x_d denote the source and destination point respectively.
- $g(x)$ where x is a point, represents the cost for reaching x from x_s . It is also referred as cost of x .
- $parent(x)$ denotes the parent of point x .

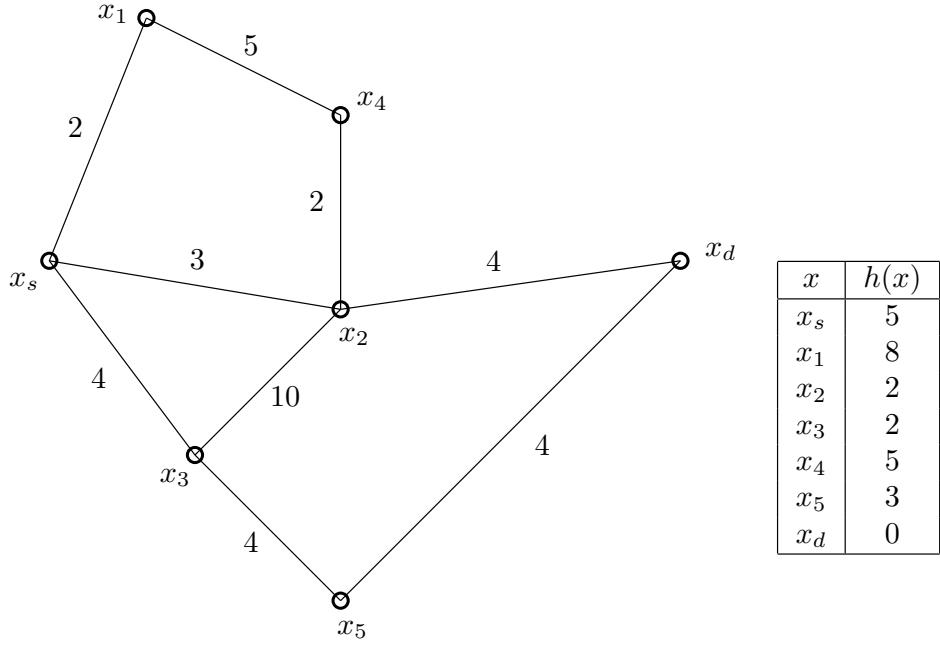


Figure 4.1: An example search problem

- $c(x_1, x_2)$ denotes the cost of connection from point x_1 to x_2 . Thus, we have $g(x) = g(\text{parent}(x)) + c(\text{parent}(x), x)$.
- $h(x)$ denotes the heuristic cost to reach from x to x_d .
- $\text{neighbours}(x)$ denotes the neighbours of point x .
- $f(x_p \rightarrow x)$ stands for the cost of point x through x_p , which is equal to $g(x_p) + c(x_p, x)$. Thus, when x is added to closed list, we have

$$g(x) = \min_{x_i \in \text{neighbours}(x)} (f(x_i \rightarrow x)) \quad (4.1)$$

$$\text{parent}(x) = \arg \min_{x_i \in \text{neighbours}(x)} (f(x_i \rightarrow x)) \quad (4.2)$$

- $d(x \rightarrow x_1, x_2)$ denotes the cost of connection from point x_1 to x_2 when x is the parent of x_1 .

4.1.1 An Example Problem

The inputs for the algorithm consist of

- x_s, x_d
- $c(x_i, x_j)$ for each edge $x_i x_j$ in the graph
- $h(x)$ for each point x in the graph

The algorithm maintains two sets of points, called the “open set” and the “closed set”. Initially, the closed set is empty while the open set contains the source point with zero cost. The cost of all other points is set to ∞ . The algorithm repeats the following steps until the destination point is added to the closed set.

$(p, g(p))$	x	$f(p \rightarrow x)$	$g(x)$	$parent(x)$	open set $\{(x, g(x) + h(x))\}$	closed set
$(x_s, 0)$	x_1	2	2	x_s	$\{(x_1, 10)\}$	$\{x_s\}$
	x_2	3	3	x_s	$\{(x_2, 5), (x_1, 10)\}$	
	x_3	4	4	x_s	$\{(x_2, 5), (x_3, 6), (x_1, 10)\}$	
$(x_2, 3)$	x_d	7	7	x_2	$\{(x_3, 6), (x_d, 7), (x_1, 10)\}$	$\{x_s, x_2\}$
	x_3	13	4	x_s	$\{(x_3, 6), (x_d, 7), (x_1, 10)\}$	
	x_4	5	5	x_2	$\{(x_3, 6), (x_d, 7), (x_1, 10), (x_4, 10)\}$	
$(x_3, 4)$	x_5	8	8	x_3	$\{(x_d, 7), (x_1, 10), (x_5, 11)\}$	$\{x_s, x_2, x_3\}$
$(x_d, 7)$	-	-	-	-	$\{(x_1, 10), (x_5, 11)\}$	$\{x_s, x_2, x_3, x_d\}$

Table 4.1: Solution to search problem from Figure 4.1 using A* search

- If the open set is empty, stop the iterations. This implies that path does not exist.
- Remove the point p with minimum value of $g(p) + h(p)$ from open set and add it to closed set.
- For each neighbour x of p that does not belong to the closed set, compute the value of $g(p) + c(p, x)$ i.e. $f(p \rightarrow x)$. If $f(p \rightarrow x) < g(x)$, set $g(x)$ to $f(p \rightarrow x)$ and $parent(x)$ to p . Add p to the open set if it has not been added already.

A point added to closed set implies that the minimum cost to reach that point from the starting point has been found. As a result, when the destination point is added to closed set, the cost of destination point denotes the minimum cost to reach it. The optimum path is determined by back-tracing. Parent of the destination point will be the previous point on the optimum path. By accessing the parent points along the chain until the starting point is found, the complete path is traced.

Table 4.1 shows the calculations which the algorithm performs for solving the example problem shown in Figure 4.1. Note that the cost for x_3 is computed twice, through parents x_s and x_2 . As $f(x_2 \rightarrow x_3) > f(x_s \rightarrow x_3)$, the parent of x_3 is set to x_s . The search terminates when x_d is added to the closed set. The parent of x_d is x_2 . Parent of x_2 is x_s . Thus, the obtained path is $x_s \rightarrow x_2 \rightarrow x_d$.

The heuristic values enable the algorithm to avoid unnecessary nodes. For example, because of high value of $h(x)$ for x_1 , its priority in open set became high because of which, the algorithm never added it to the closed set. On the other hand, because $h(x_3)$ was lesser, the algorithm chose to explore it. Thus, as long as the heuristic is admissible, changing the heuristic will not affect the path generated by the algorithm. Even if all heuristic values are set to 0, the algorithm will generate the same path, although it will have to explore additional nodes. As this chapter focuses on determining the optimum path with turn based cost functions, heuristics have not been discussed in the following sections.

4.1.2 Problems with turn based cost functions

In path planning problems, the map is divided into a grid of cells. Each cell is considered as a node on the graph and neighbouring cells are connected in the node graph with the edge cost as the cost of travelling from one cell to another.

Consider two neighbouring nodes x_1 and x_2 . For finding shortest paths, the cost function $c(x_1, x_2)$ is chosen as the distance between x_1 and x_2 . For distance, the cost from x_1 to x_2 is independent of $parent(x_1)$. Hence, once the cost of x_1 is fixed, the cost of x_2 via x_1 gets fixed. During search, if $g(x_2)$ with x_1 as parent is less than $g(x_2)$ with x_3 as parent, the minimum cost is stored and the corresponding parent is retained.

However, when considering turns, the cost between a node and its parent also depends on the parent's parent. As a result, its possible that the parent for cost greater than the minimum will lead to the globally optimum path. But this path is discarded because only the parent with minimum cost is retained. The following example demonstrates this.

Referring to Figure 4.2, consider two different costs for reaching x from w_1, w_2 . $f(w_1 \rightarrow x)$ is greater than $f(w_2 \rightarrow x)$. But as w_2, x and y are in straight line we have $d(w_1 \rightarrow x, y) < d(w_2 \rightarrow x, y)$. Consider the case where $f(w_1 \rightarrow x) + d(w_1 \rightarrow x, y) > f(w_2 \rightarrow x) + d(w_2 \rightarrow x, y)$. This means that the path to x through w_2 has less cost than the path through w_1 . Hence, it is expected that the path planning algorithm must return the path through w_2 . However, while computing the cost of x, w_2 as a parent would be discarded as the cost through w_2 is higher and hence, parent of x will be set to w_1 . This can also be seen from (4.1). Hence, the route $w_2 \rightarrow x \rightarrow y$ would never be considered in normal A* search. Because of this, the resultant path will not be optimum for a turn based criteria. In case of distance, this issue does not occur because $d(w_1 \rightarrow x, y) = d(w_2 \rightarrow x, y)$.

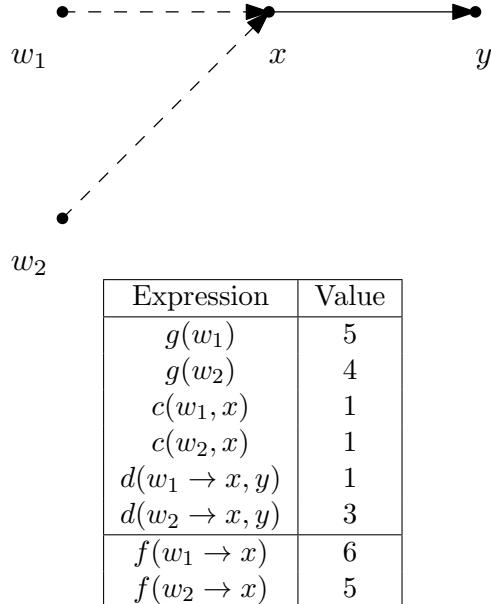


Figure 4.2: Example

As a result, turn based criteria cannot be used with normal A* search. Hence, there is a need for another approach which allows to use turn based cost functions. The Edge Based approach proposed in the next section allows the use of turn based cost functions.

4.2 The proposed Edge based A*

The normal approach considered a point as a node. In the edge based approach, an edge is considered as a node. This change allows A* search to obtain optimum results for turn based cost functions. This technique has been termed as 'Edge based A*'.

Terminology

- x_1x_2 denotes the edge with starting point x_1 and ending point x_2
- $g_e(x_1x_2)$ denotes the cost of reaching the node x_1x_2 from starting node.

- $c_e(x_1x_2, x_3x_4)$ denotes the cost of reaching the edge x_3x_4 from x_1x_2 .
- $h_e(x_1x_2)$ denotes the heuristic cost of reaching the destination from the edge x_1x_2 .
- $f_e(x_1x_2 \rightarrow x_3x_4)$ denotes the cost for reaching edge x_3x_4 through x_1x_2 . Thus,

$$f_e(x_1x_2 \rightarrow x_3x_4) = g_e(x_1x_2) + c_e(x_1x_2 \rightarrow x_3x_4).$$

Referring to example from Figure 4.2, as $d(w_2 \rightarrow x, y)$ and $d(w_1 \rightarrow x, y)$ have different costs. In edge based approach, w_1x and w_2x are different nodes. The open set will contain both nodes. When w_1x gets popped from open set, $g_e(xy) = g_e(w_1x) + c_e(w_1x, xy)$. When w_2x gets popped, $g_e(w_2x) + c_e(w_2x, xy)$ will be compared with previous $g_e(xy)$. The lesser cost will be set and the corresponding edge would be set as the parent of xy . Thus, as per our example, the route $w_2 \rightarrow x \rightarrow y$ will be chosen.

Start and end nodes

In case of normal approach (point based), there is a single start node and a single end node. When solving this problem with edge based approach, there are multiple start edges and end edges. Edges with first point as the start point are the start edges and edges with second point as the end point are the end edges. Thus, the search is stopped when any one of the end edges gets popped from the open set. Figure 4.3 shows an example of start and end nodes for a 2D grid with four point connectivity. S and D denote the source and destination points respectively.

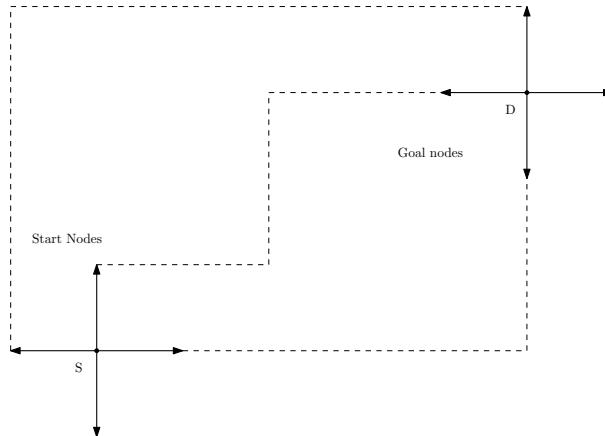


Figure 4.3: Start and End Nodes

4.2.1 Proof

Claim: When edge e is popped from open set, $g_e(e)$ is equal to the minimum cost to reach e from the starting point.

Consider the instant before edge e gets popped. As edge e is getting popped, we have

$$g_e(e) = \min_{m \in O}(g_e(m)), \quad (4.3)$$

where O denotes the set of edges/nodes in open set. P, Q denote the starting and ending points of edge e as shown in Figure 4.4. This proof uses the method of contradiction. Suppose, there exists a path from an edge $e_1 \in O$ to Q through some edge e_2 with cost less than $g_e(e)$.

$$g_e(e_2) < g_e(e). \quad (4.4)$$

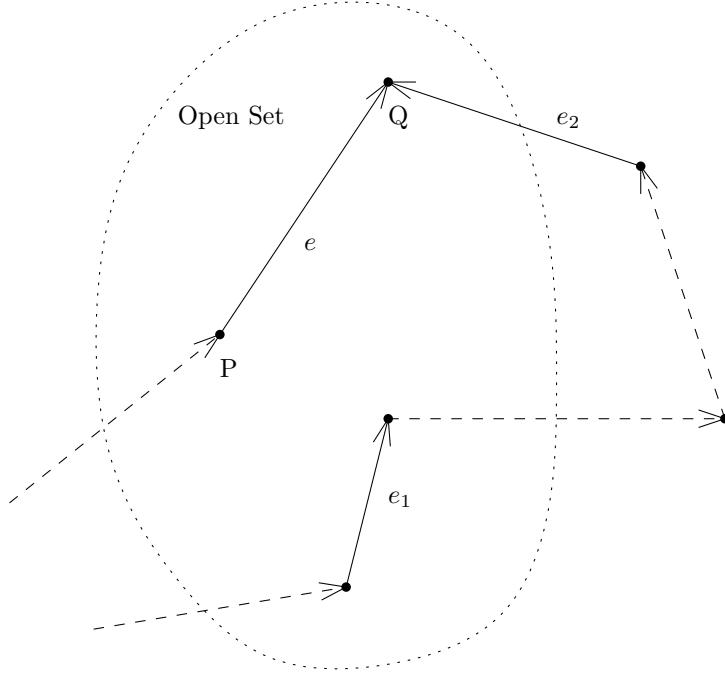


Figure 4.4: Diagram for Proof

The path till e_1 is part of the path till e_2 and the cost of a node cannot be negative. Hence, we have

$$g_e(e_1) < g_e(e_2). \quad (4.5)$$

From (4.4) and (4.5), we have

$$g_e(e_1) < g_e(e).$$

This violates our initial condition (4.3) and hence, proves that the claim is true.

Thus, when an edge ending with the goal point gets popped from the open set, we get a path with minimum cost.

4.2.2 Time Complexity

In the worst case, the search algorithms will explore all points or edges. Hence, the time complexity is proportional to number of nodes. For a grid based map of size $n \times n$, the number of edges is $2n(n - 1)$ where the number of points is n^2 . Thus, the time complexity for edge based approach and normal approach with respect to the grid size is $O[n^2]$. $2n(n - 1) \geq n^2$ for $n > 1$. Hence, the ratio of worst case computation time of edge based approach compared to the normal approach is given by

$$\frac{T(\text{edge based})}{T(\text{normal})} = 2 \left(1 - \frac{1}{n}\right).$$

4.2.3 Comparison with A*

For comparison, the following cost function has been used.

$$c_e(e_p, e) = \text{angle}(e_p, e) + 0.01 * \text{length}(e),$$

where $\text{angle}(e_1, e_2)$ denotes the magnitude of difference between directions of edges e_1 and e_2 and $\text{length}(e)$ denotes the length of the edge e . Length has been added so that the algorithm

chooses the shorter path in case there is a match. Because of the small weight to length, a path with less turns will be given priority over a shorter path. Figure 4.5 shows a comparison of paths generated for the above cost function using normal A* search and the proposed edge based A* method. The heuristic function is

$$h_e(e) = 0.01 * \text{distance}(e^2, D),$$

where e^2 denotes the endpoint of edge e , D denotes the destination point and $\text{distance}(x, y)$ denotes the distance between points x and y .

The tables 4.2 and 4.3 show the calculations performed by normal A* search and edge based A* search respectively for the search problem shown in Figure 4.5a. Each row in the table shows the computation of cost for the point/edge (e/x) through its parent point/edge (x_p/e_p) . In the notation for representing an edge, the left point denotes the starting point and the right point denotes the ending point of the edge. Thus, $(4, 3 \rightarrow 3, 3)$ denotes the edge from $(4, 3)$ to $(3, 3)$. The tables shows the order in which the algorithms explore the nodes as well as the cost computed in that iteration.

In normal A* search, the cost for the point $(3, 4)$ is calculated through two parents: $(3, 3)$ and $(4, 4)$. These represent the two paths: $(4, 3) \rightarrow (3, 3) \rightarrow (3, 4)$ and $(4, 3) \rightarrow (4, 4) \rightarrow (3, 4)$. As both the paths have a single 90° turn and as their lengths are same, the value of $g(x)$ is the same. As the heuristic is only dependent on the distance to goal, the $h(x)$ values are also same. Because the path through $(3, 3)$ is explored first, the parent of $(3, 4)$ is set to $(3, 3)$. Because these two paths are considered equivalent by the algorithm it chooses not to update the path when it explores $(4, 4)$. As a result, the algorithm produces a suboptimal path.

In edge based approach, the cost for the edge $(3, 4 \rightarrow 2, 4)$ is computed through two parent edges: $(3, 3 \rightarrow 3, 4)$ and $(4, 4 \rightarrow 3, 4)$. However, the cost of the latter option is lesser and hence the algorithm sets $(4, 4 \rightarrow 3, 4)$ as the parent of $(3, 4 \rightarrow 2, 4)$ which is the optimum choice.

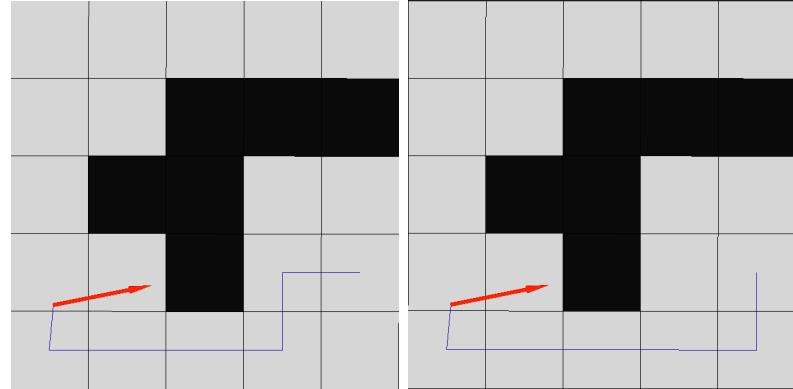
Table 4.4 shows a comparison of computation for the two approaches. The parameter ‘Nodes explored’ denotes the number of nodes for which the algorithms computed cost. In case of Example 3, the nodes explored by normal A* search were almost twice as compared to nodes explored in edge based A* search. Because a suboptimal parent is set in normal A*, the algorithm explores nodes in the incorrect direction. For example, consider the situation in example 1 (Figure 4.5a). Because the parent of $(3, 4)$ is set as $(3, 3)$, the cost of $(3, 5)$ will be less than $(2, 4)$. Hence, the algorithm will explore in the downward direction also. In case of edge based A*, because the parent edge is set optimally, nodes are not explored in the wrong direction unnecessarily.

Next, we describe the edge based concept on the existing Theta* search method Nash et al. [16].

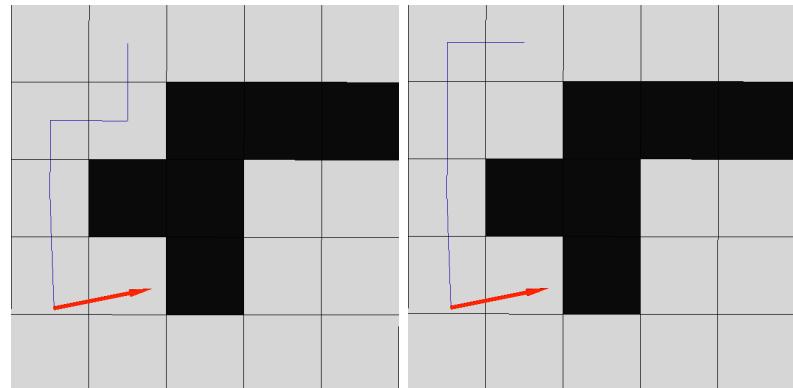
4.3 The Theta* approach

In the above sections, the edge based approach was used with A* search. This approach can be used with any search algorithm. This section focuses on the Theta* algorithm proposed by Nash et al. [16] and the application of edge based approach on Theta*. The Theta* algorithm selectively creates any-angle edges between non-adjacent points to form paths with lesser number of turns. This feature makes it suitable for generating energy efficient paths. An enhancement for the Theta* algorithm which improves the results for turn based cost functions has also been proposed.

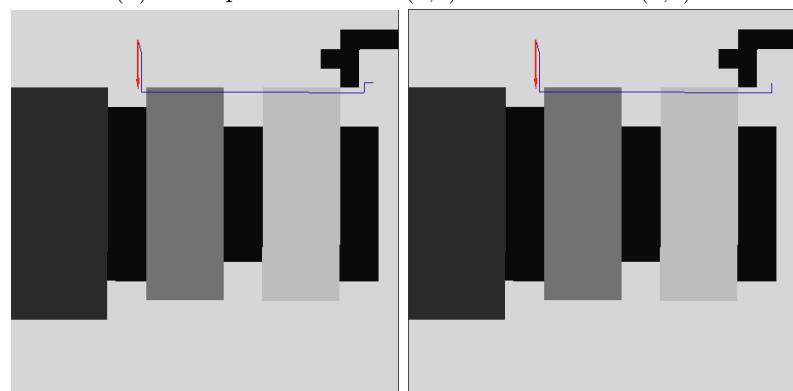
The paths in A* search are formed by connecting adjacent points. In case of Theta*, some paths between non-adjacent points are also considered during the search. When the cost for a node is computed through its parent, the cost is also computed through the parent’s parent.



(a) Example 1 - Source: (4,3) Destination: (0,3)



(b) Example 2 - Source: (1,0) Destination: (0,3)



(c) Example 3 - For a larger graph

Figure 4.5: Comparison of paths generated by Normal A* search (left) and Edge based A* search (right). Origin is on the top left corner in each grid.

x_p	x	$f(x_p \rightarrow x)$	$h(x)$	$g(x) + h(x)$
(4,3)	(3,3)	0.010000	0.030000	0.040000
(4,3)	(4,2)	0.010000	0.041231	0.051231
(4,3)	(4,4)	0.010000	0.041231	0.051231
(3,3)	(3,2)	1.590796	0.031623	1.622419
(3,3)	(3,4)	1.590796	0.031623	1.622419
(4,2)	(3,2)	1.590796	0.031623	1.622419
(4,4)	(3,4)	1.590796	0.031623	1.622419
(3,4)	(2,4)	3.171593	0.022361	3.193953
(2,4)	(1,4)	3.181593	0.014142	3.195735
(1,4)	(0,4)	3.191593	0.010000	3.201593
(1,4)	(1,3)	4.762389	0.010000	4.772389
(0,4)	(0,3)	4.772389	0.000000	4.772389
(1,3)	(0,3)	6.343185	0.000000	6.343185

Table 4.2: Normal A* calculations for Example 1 (Figure 4.5a)

e_p	e	$f_e(e_p \rightarrow e)$	$h_e(e)$	$g_e(e) + h_e(e)$
-	(4,3 → 3,3)	0.010000	0.030000	0.040000
-	(4,3 → 4,2)	0.010000	0.041231	0.051231
-	(4,3 → 4,4)	0.010000	0.041231	0.051231
(4,3 → 3,3)	(3,3 → 3,2)	1.590796	0.031623	1.622419
(4,3 → 3,3)	(3,3 → 3,4)	1.590796	0.031623	1.622419
(4,3 → 4,2)	(4,2 → 3,2)	1.590796	0.031623	1.622419
(4,3 → 4,4)	(4,4 → 3,4)	1.590796	0.031623	1.622419
(3,3 → 3,4)	(3,4 → 2,4)	3.171593	0.022361	3.193953
(3,3 → 3,4)	(3,4 → 4,4)	3.171593	0.041231	3.212824
(4,4 → 3,4)	(3,4 → 2,4)	1.600796	0.022361	1.623157
(4,4 → 3,4)	(3,4 → 3,3)	3.171593	0.030000	3.201593
(3,3 → 3,2)	(3,2 → 4,2)	3.171593	0.041231	3.212824
(4,2 → 3,2)	(3,2 → 3,3)	3.171593	0.030000	3.201593
(3,4 → 2,4)	(2,4 → 1,4)	1.610796	0.014142	1.624938
(3,4 → 2,4)	(2,4 → 1,4)	1.610796	0.014142	1.624938
(2,4 → 1,4)	(1,4 → 0,4)	1.620796	0.010000	1.630796
(1,4 → 0,4)	(0,4 → 0,3)	3.201593	0.000000	3.201593
(3,4 → 3,3)	(3,3 → 4,3)	4.752389	0.040000	4.792389
(3,2 → 3,3)	(3,3 → 4,3)	4.752389	0.040000	4.792389

Table 4.3: Edge based A* calculations for Example 1 (Figure 4.5a)

	Nodes explored		Search completion time (ms)	
	Normal A*	Edge based A*	Normal A*	Edge based A*
Example 1	13	19	0	0
Example 2	9	13	0	0
Example 3	2137	1056	60	30

Table 4.4: Computation comparison of Normal A* and Edge based A* methods

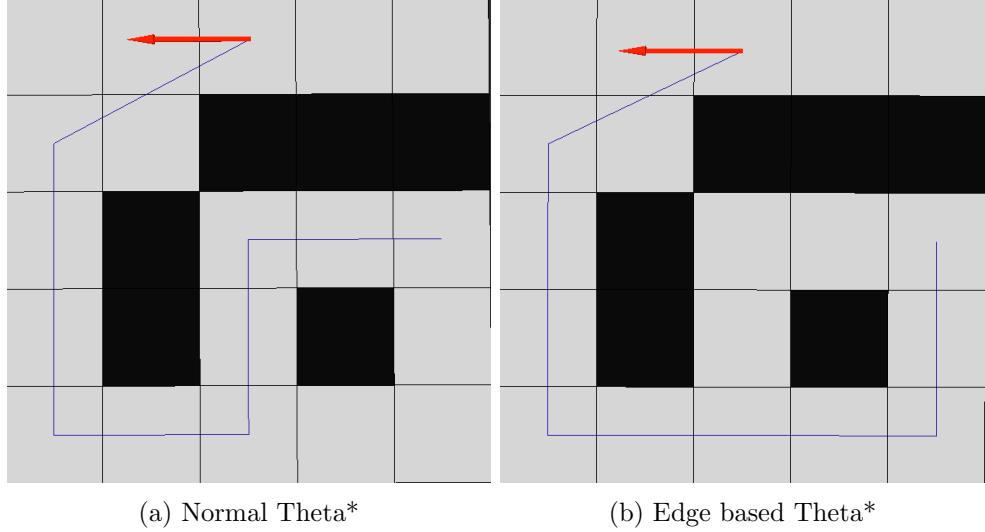


Figure 4.6: Paths generated by Normal Theta* and Edge based Theta*

If the latter cost is lesser, the parent of the node is set to the previous parent's parent. Thus, the node which was originally the parent gets removed from the path. Algorithm 4.1 shows the pseudo code for the theta star method.

The next section describes the issues with Theta* and how the proposed Edge based Theta* algorithm solves them.

4.3.1 The proposed Edge based Theta*

The difference between A* and Theta* lies only in the *UpdateNode* procedure from Algorithm 4.1. From the perspective of turn based cost functions, the Theta* technique also has the limitations described in Section 4.1.2. Figure 4.6a shows an example of suboptimal path generated by Theta*. In this example, cost for the node (2,4) through the parents (2,2) and (4,4) are equal (Table 4.5). Hence, the parent of (2,4) is set to (2,2) which ultimately leads to the suboptimal path.

In case of Edge based Theta*, the cost for $(2, 4 \rightarrow 1, 4)$ is lesser through the parent $(4, 3 \rightarrow 4, 4)$ than through $(2, 3 \rightarrow 2, 4)$. As a result, the optimum path is obtained as shown in Figure 4.6b.

4.3.2 Further enhancement on the proposed Edge based Theta*

The Theta* algorithm assumes that if a node and its parent's parent node have line of sight, the cost through parent's parent will always be optimum. Referring to the example in Figure 4.6, $(3, 2 \rightarrow 2, 2)$ is the parent of $(2, 2 \rightarrow 2, 3)$. Now, when the cost for $(2, 3 \rightarrow 2, 4)$ is computed through $(2, 2 \rightarrow 2, 3)$, the parent of $(2, 3 \rightarrow 2, 4)$ is set as $(3, 2 \rightarrow 2, 2)$ as there is no obstacle between $(2, 2)$ and $(2, 4)$. When cost of $(2, 4 \rightarrow 1, 4)$ is computed through $(2, 3 \rightarrow 2, 4)$, the line of sight between $(2, 2)$ and $(1, 4)$ is checked and it returns false. However, had the parent of $(2, 3 \rightarrow 2, 4)$ been $(2, 2 \rightarrow 2, 3)$, the line of sight check between $(1, 4)$ and $(2, 3)$ would have returned true and a better path could have been obtained. Thus, in this case, having a short path through the parent's parent caused the resultant path to be less optimum which does not happen in case of distance based cost function. The updated procedure for *UpdateNode* in Algorithm 4.2 prevents these cases resulting in a better path. The results of this method have been shown in Figure 4.7b. The optimum path through this method has cost 4.017701

Algorithm 4.1 Theta* Algorithm

```
function THETASTAR(start, goal)
  x  $\leftarrow$  start
  while x  $\neq$  goal do
    for all n  $\in$  neighbours(x) do
      if n  $\notin$  closedSet then
        UPDATERNODE(x, n)
      end if
    end for
    if openQueue is empty then
      return nil
    end if
    x  $\leftarrow$  pop from openQueue
    push x to closedSet
  end while
  if x  $\neq$  goal then
    return nil
  end if
  initialize a list path add x to path
  repeat
    x  $\leftarrow$  Parent(x)
    add x to path
  until x  $\neq$  start
  return path
end function

procedure UPDATERNODE(parentnode, node)
  if LINEOFSIGHT(Parent(parentnode), node) then
    g  $\leftarrow$  COMPUTECOST(Parent(parentnode), node)
    if g  $<$  G(node) then
      G(node)  $\leftarrow$  g
      Parent(node)  $\leftarrow$  Parent(parentnode)
      if node  $\notin$  openQueue then
        push node to openQueue
      end if
    end if
  end if
  else
    g  $\leftarrow$  COMPUTECOST(parentnode, node)
    if g  $<$  G(node) then
      G(node)  $\leftarrow$  g
      Parent(node)  $\leftarrow$  parentnode
      if node  $\notin$  openQueue then
        push node to openQueue
      end if
    end if
  end if
end if
end procedure
```

x_p	x	$f(x_p \rightarrow x)$	$h(x)$	$g(x) + h(x)$
(4,2)	(3,2)	0.010000	0.020000	0.030000
(4,2)	(4,3)	0.010000	0.031623	0.041623
(4,2)	(2,2)	0.020000	0.022361	0.042361
(4,2)	(4,4)	0.020000	0.041231	0.061231
(2,2)	(2,3)	1.600796	0.031623	1.632419
(4,4)	(3,4)	1.600796	0.040000	1.640796
(2,2)	(2,4)	1.610796	0.041231	1.652027
(4,4)	(2,4)	1.610796	0.041231	1.652027
(2,4)	(1,4)	3.191593	0.044721	3.236314
(2,4)	(0,4)	3.201593	0.050000	3.251593
(0,4)	(0,3)	4.782389	0.042426	4.824815
(0,4)	(0,2)	4.792389	0.036056	4.828444
(0,4)	(0,1)	4.802389	0.031623	4.834012
(0,4)	(0,0)	4.812389	0.030000	4.842389
(0,1)	(1,1)	6.383185	0.022361	6.405546
(0,0)	(1,0)	6.393185	0.020000	6.413185
(0,1)	(1,0)	5.601929	0.020000	5.621929
(0,1)	(2,0)	5.931898	0.010000	5.941898
(0,1)	(2,0)	5.931898	0.010000	5.941898
(0,1)	(3,0)	6.083058	0.000000	6.083058

Table 4.5: Calculations for Theta* algorithm for example in Figure 4.6a

(Table 4.7) compared to 4.512261 for edge based Theta*.

4.4 Improvement of energy saving using Edge based approaches

The primary motivation for the Edge based approach is to enable use of cost functions which involve turns. Energy consumption of a robot is one such cost function. The cost function is based on the energy model discussed in Section 3.1.2. For estimating the energy consumption, the robot is assumed to stop when there is a turn. The angular velocity during turning and the linear velocity while travelling between waypoints are constant and are denoted by ω and v respectively. Thus, at a turn, the cost will be equal to the sum of the energy required to turn and the energy required to accelerate the robot. Algorithm 4.3 shows the pseudo code for computing cost for a node.

In Chapter 3, the friction co-efficient of a point was used in the energy cost function. In case of A*, the parent and child nodes are neighbours of each other. As a result, the average friction co-efficient between the parent and child nodes was used. In case of Theta*, the parent and the child nodes need not be neighbours, though the line of sight check ensures that there is no obstacle on the line joining them. As a result, for computing the friction co-efficient, the line joining the nodes is split into small parts and the average friction co-efficient for those parts is used for energy cost calculation.

The next section compares the paths generated by different search methods discussed in this report with respect to their energy saving.

e_p	e	$f_e(e_p \rightarrow e)$	$h_e(e)$	$g_e(e) + h_e(e)$
(4,2)	(4,2 → 3,2)	0.010000	0.020000	0.030000
(4,2)	(4,2 → 4,3)	0.010000	0.031623	0.041623
(4,2)	(3,2 → 2,2)	0.020000	0.022361	0.042361
(4,2)	(4,3 → 4,4)	0.020000	0.041231	0.061231
(3,2 → 2,2)	(2,2 → 2,3)	1.600796	0.031623	1.632419
(4,3 → 4,4)	(4,4 → 3,4)	1.600796	0.040000	1.640796
(3,2 → 2,2)	(2,3 → 2,4)	1.610796	0.041231	1.652027
(4,3 → 4,4)	(3,4 → 2,4)	1.610796	0.041231	1.652027
(2,3 → 2,4)	(2,4 → 1,4)	3.191593	0.044721	3.236314
(2,3 → 2,4)	(2,4 → 3,4)	3.191593	0.040000	3.231593
(4,3 → 4,4)	(2,4 → 1,4)	1.620796	0.044721	1.665518
(3,4 → 2,4)	(2,4 → 2,3)	3.191593	0.031623	3.223215
(4,3 → 4,4)	(1,4 → 0,4)	1.630796	0.050000	1.680796
(4,3 → 4,4)	(1,4 → 0,4)	1.630796	0.050000	1.680796
(1,4 → 0,4)	(0,4 → 0,3)	3.211593	0.042426	3.254019
(3,4 → 2,4)	(2,3 → 2,2)	3.201593	0.022361	3.223953
(2,3 → 2,2)	(2,2 → 3,2)	4.782389	0.020000	4.802389
(2,3 → 2,4)	(3,4 → 4,4)	3.201593	0.041231	3.242824
(3,4 → 4,4)	(4,4 → 4,3)	4.782389	0.031623	4.814012
(1,4 → 0,4)	(0,3 → 0,2)	3.221593	0.036056	3.257648
(1,4 → 0,4)	(0,2 → 0,1)	3.231593	0.031623	3.263215
(1,4 → 0,4)	(0,1 → 0,0)	3.241593	0.030000	3.271593
(0,2 → 0,1)	(0,1 → 1,1)	4.812389	0.022361	4.834750
(0,1 → 0,0)	(0,0 → 1,0)	4.822389	0.020000	4.842389
(2,3 → 2,2)	(3,2 → 4,2)	4.792389	0.022361	4.814750
(3,4 → 4,4)	(4,3 → 4,2)	4.792389	0.022361	4.814750
(0,2 → 0,1)	(1,1 → 1,0)	4.031133	0.020000	4.051133
(0,2 → 0,1)	(1,0 → 0,0)	3.241593	0.030000	3.271593
(0,2 → 0,1)	(1,0 → 2,0)	4.361102	0.010000	4.371102
(0,2 → 0,1)	(0,0 → 0,1)	6.373185	0.031623	6.404808
(0,2 → 0,1)	(2,0 → 3,0)	4.512261	0.000000	4.512261

Table 4.6: Calculations for Theta* algorithm for example in Figure 4.6a

Algorithm 4.2 Theta* Enhancement

```

procedure UPDATERNODE(parentnode, node)
  g0  $\leftarrow$  COMPUTECOST(parentnode, node)
  if LineOfSight(node, Parent(parentnode)) then
    g  $\leftarrow$  COMPUTECOST(Parent(parentnode), node)
    if g < g0 and g < G(node) then ▷ Use this option only If cost is lesser
      G(node)  $\leftarrow$  g
      Parent(node)  $\leftarrow$  Parent(parentnode)
      if node  $\notin$  openQueue then
        push node to openQueue
      end if
    else if g0 < G(node) then
      G(node)  $\leftarrow$  g0
      Parent(node)  $\leftarrow$  parentnode
      if node  $\notin$  openQueue then
        push node to openQueue
      end if
    end if
  else
    if g0 < G(node) then
      G(node)  $\leftarrow$  g0
      Parent(node)  $\leftarrow$  parentnode
      if node  $\notin$  openQueue then
        push node to openQueue
      end if
    end if
  end if
end procedure

```

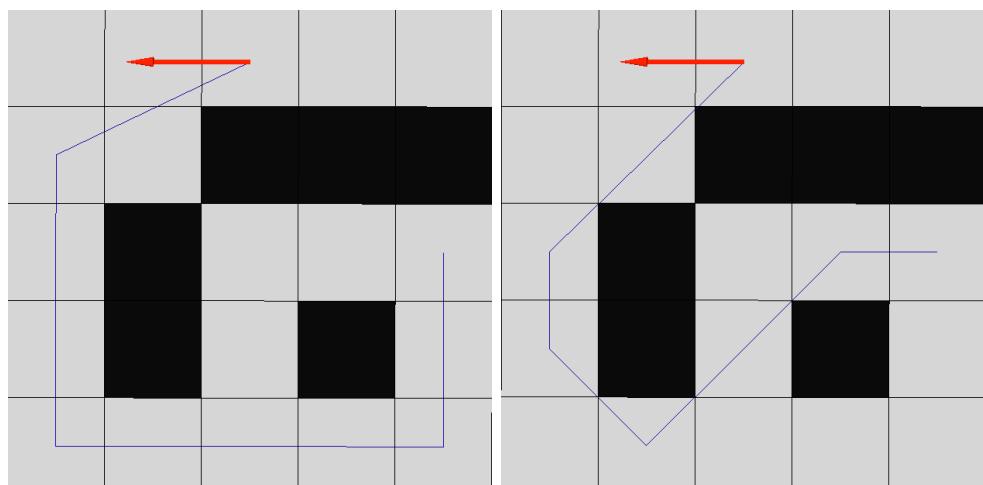


Figure 4.7: Paths generated by Edge based Theta* and Enhanced Theta*

e_p	e	$f_e(e \rightarrow e)$	$h_e(e)$	$g_e(e) + h_e(e)$
(4,2)	(4,2 → 3,2)	0.010000	0.022361	0.032361
(4,2)	(4,2 → 4,3)	0.010000	0.036056	0.046056
(4,2 → 3,2)	(3,2 → 2,2)	0.020000	0.020000	0.040000
(4,2)	(3,2 → 2,2)	0.020000	0.020000	0.040000
(3,2 → 2,2)	(2,2 → 2,3)	1.600796	0.030000	1.630796
(4,2 → 3,2)	(2,2 → 2,3)	0.809540	0.030000	0.839540
(4,2 → 4,3)	(4,3 → 4,4)	0.020000	0.044721	0.064721
(4,2)	(4,3 → 4,4)	0.020000	0.044721	0.064721
(4,3 → 4,4)	(4,4 → 3,4)	1.600796	0.041231	1.642027
(4,2 → 4,3)	(4,4 → 3,4)	0.809540	0.041231	0.850771
(2,2 → 2,3)	(2,3 → 2,4)	1.604938	0.040000	1.644938
(4,4 → 3,4)	(3,4 → 2,4)	1.604938	0.040000	1.644938
(2,3 → 2,4)	(2,4 → 1,4)	3.185735	0.041231	3.226966
(2,2 → 2,3)	(2,4 → 1,4)	0.823682	0.041231	0.864913
(2,3 → 2,4)	(2,4 → 3,4)	3.185735	0.041231	3.226966
(2,2 → 2,3)	(2,4 → 3,4)	2.394479	0.041231	2.435710
(2,4 → 1,4)	(1,4 → 0,4)	1.619081	0.044721	1.663802
(3,4 → 2,4)	(2,4 → 2,3)	3.185735	0.030000	3.215735
(4,4 → 3,4)	(2,4 → 2,3)	2.394479	0.030000	2.424479
(1,4 → 0,4)	(0,4 → 0,3)	3.199877	0.036056	3.235932
(2,4 → 1,4)	(0,4 → 0,3)	2.408621	0.036056	2.444676
(2,4 → 2,3)	(2,3 → 2,2)	3.189877	0.020000	3.209877
(2,4 → 3,4)	(3,4 → 4,4)	3.189877	0.044721	3.234598
(0,4 → 0,3)	(0,3 → 0,2)	3.204019	0.028284	3.232303
(2,3 → 2,2)	(2,2 → 3,2)	4.770673	0.022361	4.793034
(2,4 → 2,3)	(2,2 → 3,2)	3.979417	0.022361	4.001778
(0,3 → 0,2)	(0,2 → 0,1)	3.214019	0.022361	3.236380
(0,4 → 0,3)	(0,2 → 0,1)	3.214019	0.022361	3.236380
(3,4 → 4,4)	(4,4 → 4,3)	4.770673	0.036056	4.806729
(2,4 → 3,4)	(4,4 → 4,3)	3.979417	0.036056	4.015473
(0,2 → 0,1)	(0,1 → 0,0)	3.224019	0.020000	3.244019
(0,3 → 0,2)	(0,1 → 0,0)	3.224019	0.020000	3.244019
(0,2 → 0,1)	(0,1 → 1,1)	4.794815	0.014142	4.808958
(0,3 → 0,2)	(0,1 → 1,1)	4.003559	0.014142	4.017701
(0,1 → 0,0)	(0,0 → 1,0)	4.804815	0.010000	4.814815
(0,2 → 0,1)	(0,0 → 1,0)	4.013559	0.010000	4.023559
(2,2 → 3,2)	(3,2 → 4,2)	4.774815	0.028284	4.803100
(4,4 → 4,3)	(4,3 → 4,2)	4.774815	0.028284	4.803100
(0,1 → 1,1)	(1,1 → 1,0)	4.798958	0.010000	4.808958
(0,3 → 0,2)	(1,1 → 1,0)	3.690027	0.010000	3.700027
(1,1 → 1,0)	(1,0 → 0,0)	5.734471	0.020000	5.754471
(0,3 → 0,2)	(1,0 → 0,0)	3.224019	0.020000	3.244019
(1,1 → 1,0)	(1,0 → 2,0)	4.807176	0.000000	4.807176
(0,3 → 0,2)	(1,0 → 2,0)	4.017701	0.000000	4.017701
(1,0 → 0,0)	(0,0 → 0,1)	6.375612	0.022361	6.397972
(0,3 → 0,2)	(0,0 → 0,1)	3.214019	0.022361	3.236380
(0,0 → 0,1)	(0,1 → 0,2)	6.365612	0.028284	6.393896
(0,3 → 0,2)	(0,1 → 0,2)	6.345612	0.028284	6.373896

Table 4.7: Calculations for Enhanced Theta* algorithm for example in Figure 4.7b

Algorithm 4.3 Energy based cost function for edge based search

Require: width: B , mass: M , Inertia: I , Non-motor Power consumption: P_{other} , linear velocity: v , angular velocity: ω

function COMPUTECOST(*parentEdge*, *edge*)

$\theta \leftarrow \text{angle}(\text{parentEdge}, \text{edge})$ ▷ Direction change in $[0, \pi]$

$e_{turn} \leftarrow 0$

$time \leftarrow 0$

$\mu \leftarrow \text{averageFrictionBetweenEdges}(\text{parentEdge}, \text{edge})$

if $\theta > 0$ **then**

$e_{f,turn} \leftarrow \theta * B * \mu * M * 9.81$

$e_{k,turn} \leftarrow \frac{1}{2}I(\omega)^2 + \frac{1}{2} * M * v^2$

$e_{turn} \leftarrow e_{f,turn} + e_{k,turn}$

$time \leftarrow time + (\theta/\omega)$

end if

$e_{res} \leftarrow 2 * \mu * M * 9.81 * \text{length}(\text{edge})$

$time \leftarrow time + \text{length}(\text{edge})/v$

$e_{other} \leftarrow P_{other} * time$

return $G(\text{parentEdge}) + (e_{turn} + e_{res})$

end function

4.4.1 Comparison of search algorithms

The following four methods have been compared for their energy savings:

1. A* method
2. Theta* method
3. Edge based A* method
4. Edge based Theta* method

As the A* and Theta* methods do not support the turn based cost function proposed in Algorithm 4.3, the friction based cost function from Section 3.1.3 has been used. The original cost function involved a penalty factor ρ which caused the astar path to be away from the obstacles. However, this would influence the optimality of the path in terms of energy and hence, for the purpose of comparing energy profiles, the ρ factor is not used. The cost function used for A* and Theta* is given by

$$c(x_p, x) = 2\mu_{x_p, x} mg \text{ distance}(x_p, D),$$

where D is the destination point, m is the mass of the robot, g is the acceleration due to gravity.

Algorithm 4.3 is used to compute the energy profiles for the paths generated by the four methods. The algorithms have been executed several times using random source and destination points. The trajectories for the best case with maximum energy savings for Edge based Theta* have been shown in Figure 4.8 and the energy profile comparison have been shown in Table 4.8. Figure 4.10 and Table 4.9 show the same for the worst case scenario with minimum savings. Theoretically, the worst case savings can never be greater than 0% as the algorithms will produce exactly same paths when the source and destination lie on a straight line parallel to either axes of the grid. The histograms in Figure 4.9 show the frequencies of energy savings for different methods as well as their average savings.

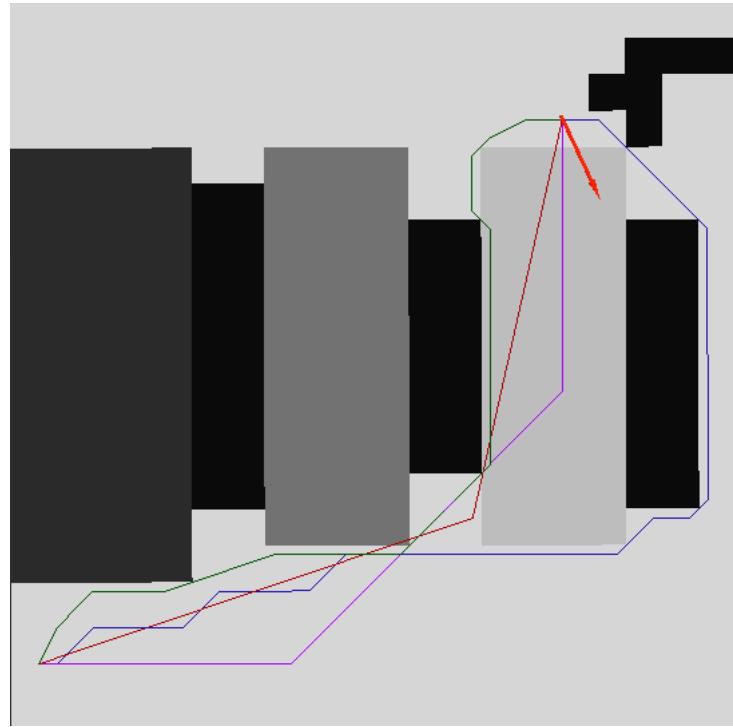


Figure 4.8: Maximum energy savings for Edge based Theta*

Method	Savings	E_{total}	$E_{ke,linear}$	E_{turns}	E_{res}	E_{other}	$length(m)$	$time(s)$
A* (Blue)	0%	2752	1106.08	474.37	584.60	586.65	66.21	33.14
Theta* (Green)	24%	2090	682.08	433.52	509.55	464.56	52.52	26.25
Edge A* (Purple)	42%	1598	571.47	79.29	506.47	440.39	50.21	24.88
Edge Theta* (Red)	64%	990	36.87	39.78	494.09	419.36	47.86	23.69

Table 4.8: Energy profile comparison for best case scenario in Figure 4.8 (All energy values are in Joules and savings are calculated with respect to A*)

Method	Savings	E_{total}	$E_{ke,linear}$	E_{turns}	E_{res}	E_{other}	$length(m)$	$time(s)$
A* (Blue)	0.00%	83	0.00	0.00	48.03	34.98	4.00	1.98
Theta* (Green)	0.00%	83	0.00	0.00	48.03	34.98	4.00	1.98
Edge A* (Purple)	0.00%	83	0.00	0.00	48.03	34.98	4.00	1.98
Edge Theta* (Red)	0.00%	83	0.00	0.00	48.03	34.98	4.00	1.98

Table 4.9: Energy profile comparison for worst case scenario in Figure 4.10 (All energy values are in Joules and savings are calculated with respect to A*)

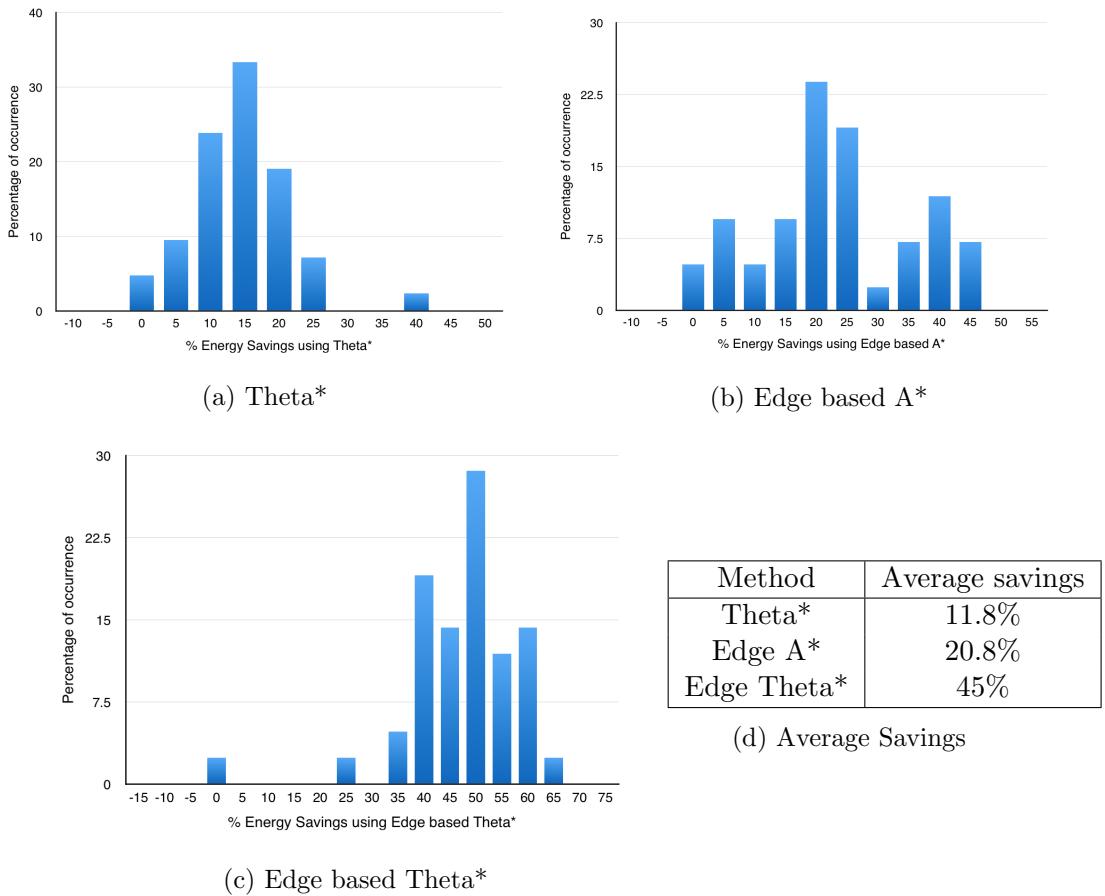


Figure 4.9: Histograms showing frequency of energy savings for different methods and average savings compared with A*

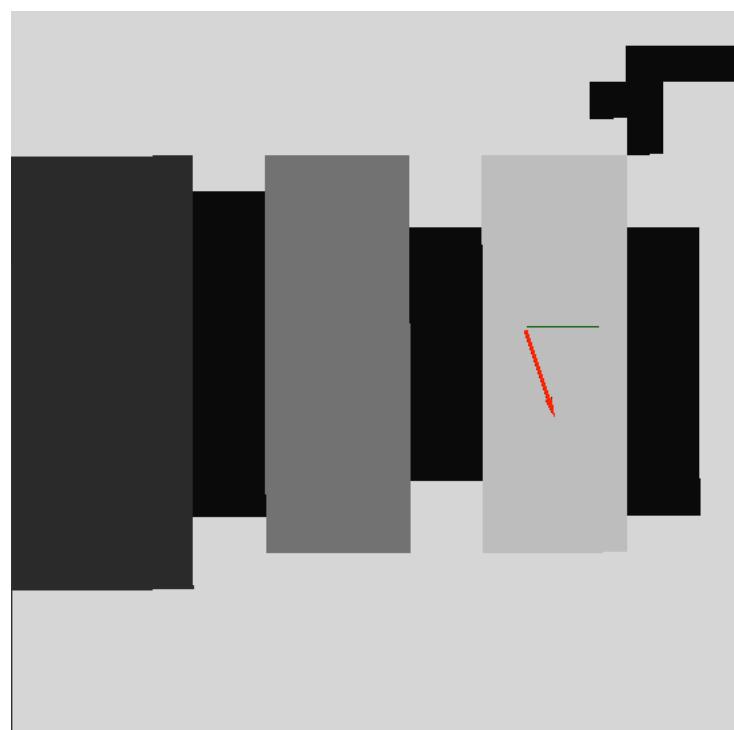


Figure 4.10: Minimum energy savings for Edge based Theta* (All paths are identical)

Chapter 5

Conclusions

Chapter 3 proposed a new method for generating trajectories using Dubins paths. Figure 5.1 shows the probability distribution of percentage energy savings compared to the Bézier curves method proposed by Liu and Sun [3]. The proposed method computes energy efficient trajectories in linear time complexity with respect to the number of waypoints compared to the exponential complexity of the technique using Bézier curves.

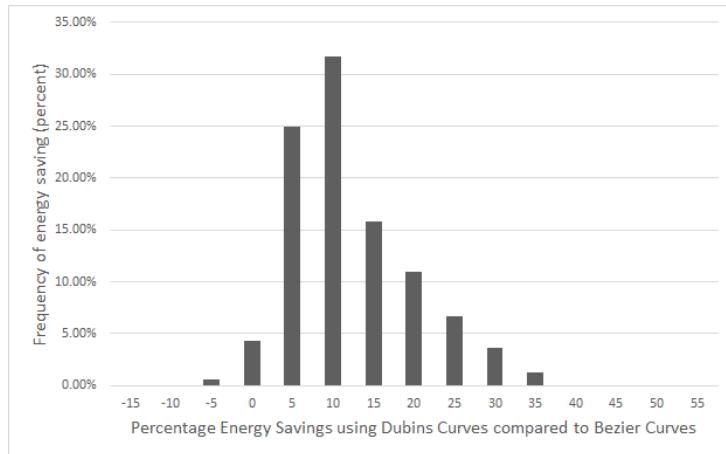


Figure 5.1: Probability distribution of percentage energy savings using A* with Dubins path compared to A* with Bézier curves (Average savings: 14.8%)

Chapter 4 explained why conventional A* search method results in a suboptimal path when used with a turn based cost function. The Edge based A* and Edge based Theta* methods were proposed to allow use of turn based cost function. The histograms in Figure 4.9 show the probability distribution of percent energy savings of Edge based A*, Edge based Theta* and Theta* methods when compared to A* method. For the comparison in Figure 4.9, energy consumption estimates were used as the continuous trajectories were unknown. In this chapter, these methods are used with the proposed trajectory planners for comparing the energy consumption.

This chapter draws out conclusions through comparison of energy savings when different combinations of waypoint planners and trajectory planners are used together.

5.1 Impact of trajectory planner

Figure 5.1 shows the comparison of Dubins trajectory planner with the Bézier curves trajectory planner keeping the waypoints planner same (A*). Figure 5.2a shows the comparison of Edge

based Theta* with A* keeping the trajectory planner same (Bézier curves). Thus, the first comparison shows the effect of improving the trajectory planner, while the second comparison shows the effect of improving the waypoints planner keeping the baseline as A* with Bézier curves. The average savings in the first case are 14.8% while the range of savings is [-5%, 35%]. In the second case, the average savings are 32.87% and the range of savings is [5%, 65%]. The higher averages and higher range denotes that the impact of waypoints planning is more than trajectory planning.

5.2 Impact of waypoints planner

Edge based Theta* has two improvements over A*. Firstly, it supports cost function involving turns and hence, uses a better energy consumption estimate. Use of Theta* supports any-angle paths which reduces the number of turns and thus, reduces the energy consumption. Edge based A* includes only the first improvement. Figure 5.2b shows the comparison of Edge based A* compared to A* keeping the trajectory planner same (Bézier curves). Figure 5.2a shows the improvement in energy savings using Edge based Theta* compared to A* for Bézier curves. The comparison shows that the improvement in cost function has contributed more to energy savings compared to the any-angle path functionality.

On the other hand, Figure 5.2c demonstrates the impact of any-angle paths on energy savings by comparing Edge based Theta* with Edge based A*. The average savings in this case are 6.52% with a range of [0,25%]. Average savings for improvement in cost function through the use of Edge based approach are 26.44% with a range of [0,55%]. This confirms the conclusion that the cost function for searches have more impact on energy savings than the search algorithm.

5.3 Conclusions

The research questions posed in Chapter 2 can now be answered based on the results.

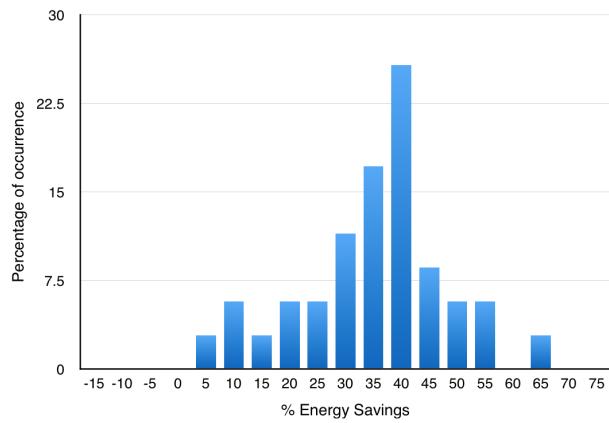
- *Can the expensive algorithms be simplified in order to reduce the computational complexity while still retaining the energy consumption optimality?*

The proposed trajectory planner using Dubins paths has linear time complexity with respect to number of waypoints. The existing method which makes use of Bézier curves has exponential complexity. Apart from being less expensive computationally, the proposed method also provides higher savings.

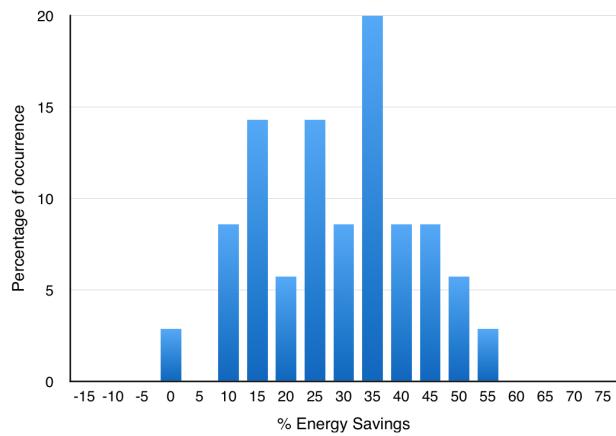
- *Can an energy consumption criteria which considers the turns in the path be used at the waypoints planning stage to generate waypoints which will lead to minimum energy consumption?*

The existing method for waypoints planning A* search results in suboptimal paths with a turn based cost function. The Edge based waypoints planning method has been proposed that generates optimum results using turn based cost function. The application of this method for energy optimal path planning has provided average energy savings of 32.87%.

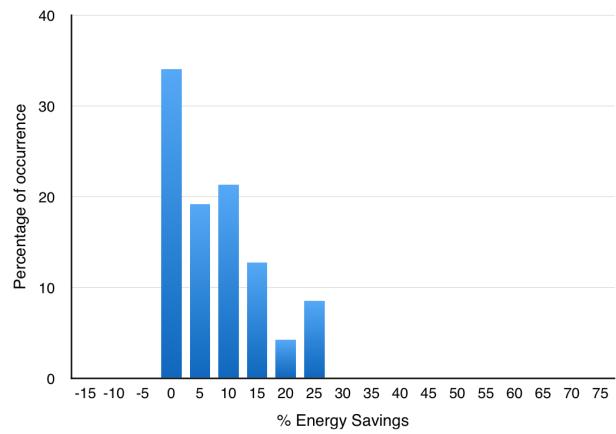
- *Which aspect among waypoints planning and trajectory planning has more impact on energy consumption?* The analysis discussed in Sections 5.1 and 5.2 concludes that waypoints planning has more impact on energy savings compared to trajectory planning.



(a) Edge based Theta* compared to A* for Bézier curves
(Average savings: 32.87%)



(b) Edge based A* compared to A* for Bézier curves (Average savings: 26.44%)



(c) Edge based Theta* compared to Edge based A* for Bézier curves (Average savings: 6.52%)

Figure 5.2: Energy Savings comparison using different combinations of proposed methods

Appendix A

Calculating Dubins path

Consider the Dubins path computation between two waypoints q_1, q_2 as shown in Figure A.1.

- $d(p_1, p_2)$ denotes the distance between two points p_1, p_2 .
- r_0 is the radius of the two circles.
- c_1 and c_2 are the centers of the two circles touching q_1 and q_2 respectively.
- D_{c12} is the distance between the two circle centers c_1 and c_2 .
- θ_1 and θ_2 are the directions of the waypoints q_1 and q_2 respectively.
- θ_{r1} and θ_{r2} are the central angles corresponding to the circular arcs in the two circles which are part of the path.
- $Left$ indicates counterclockwise direction (+ve) and $right$ indicates clockwise direction(-ve).
- $\Delta_L(\theta_1, \theta_2)$ denotes the amount of rotation in *left* direction required to reach from θ_1 to θ_2 . The amount of rotation will always lie in the range $[0, 2\pi]$.
- $\Delta_R(\theta_1, \theta_2)$ is similar to $\Delta_L(\theta_1, \theta_2)$ except that it denotes the difference in the *right* direction.

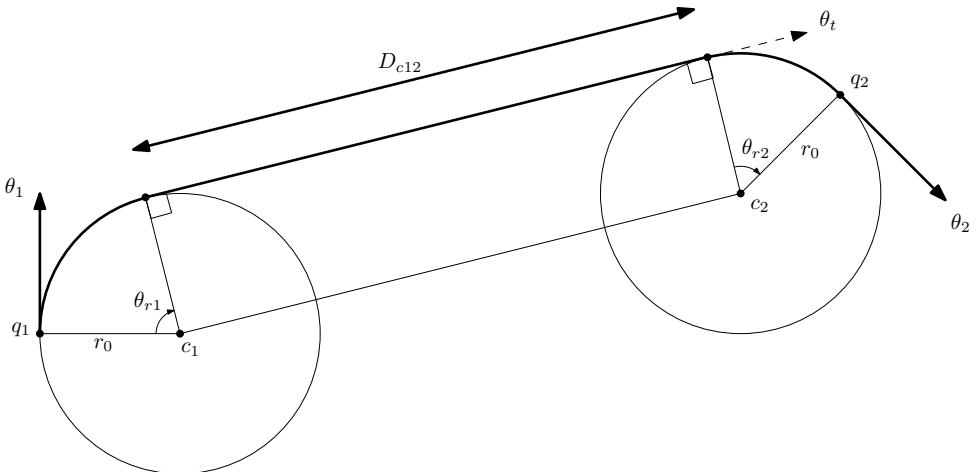


Figure A.1: An RSR trajectory

Thus, referring to figure A.1, we get

$$\theta_{r1} = \Delta_R(\theta_1, \theta_t) \quad \text{and} \quad \theta_{r2} = \Delta_L(\theta_2, \theta_t).$$

A.1 Common Calculations

Determining circle centers

The direction of the vector from the starting point towards the center of the circle will depend on the turn direction.

$$\theta_{1c} = \begin{cases} \theta_1 - \pi/2 & \text{left turn,} \\ \theta_1 + \pi/2 & \text{right turn,} \end{cases}$$

$$\therefore c_1 = (x_1 + r_1 \cos \theta_{1c}, y_1 + r_1 \sin \theta_{1c}).$$

Similarly, c_2 can also be calculated.

Vector between circle centers

$$D_{c12} = \sqrt{(c_{1x} - c_{2x})^2 + (c_{1y} - c_{2y})^2}$$

$$\therefore D_{c12} = \sqrt{(r_1 \cos \theta_{1c} - r_2 \cos \theta_{2c} + x_1 - x_2)^2 + (r_1 \sin \theta_{1c} - r_2 \sin \theta_{2c} + y_1 - y_2)^2}.$$

The direction of the vector from c_1 to c_2 can be calculated as

$$\theta_{c12} = \arctan \left(\frac{c_{2y} - c_{1y}}{c_{2x} - c_{1x}} \right).$$

A.2 Determining Path Length & Critical Values

Critical values are those values of the circle radii at which either the length of the trajectory is discontinuous or the type of the shortest trajectory changes. Determining critical values is essential to determine the search space for the optimization problem.

A.2.1 RSR

Consider the radii of the circles at c_1, c_2 to be r_1, r_2 as shown in figure A.2. For tangent to exist, a required condition is $D_{c12} > |r_1 - r_2|$ where D_{c12} is the distance between two centers. Then, we have the following equations.

$$L_t = \sqrt{D_{c12}^2 - (r_2 - r_1)^2}$$

$$\therefore \alpha = \arccos \left(\frac{\sqrt{D_{c12}^2 - (r_1 - r_2)^2}}{D_{c12}} \right).$$

Calculation of θ_t depends on which among r_1, r_2 is greater. If θ_{c12} is the direction of the vector from c_1 to c_2 , we get

$$\theta_t = \begin{cases} \theta_{c12} + \alpha & r_1 < r_2, \\ \theta_{c12} - \alpha & r_1 > r_2. \end{cases}$$

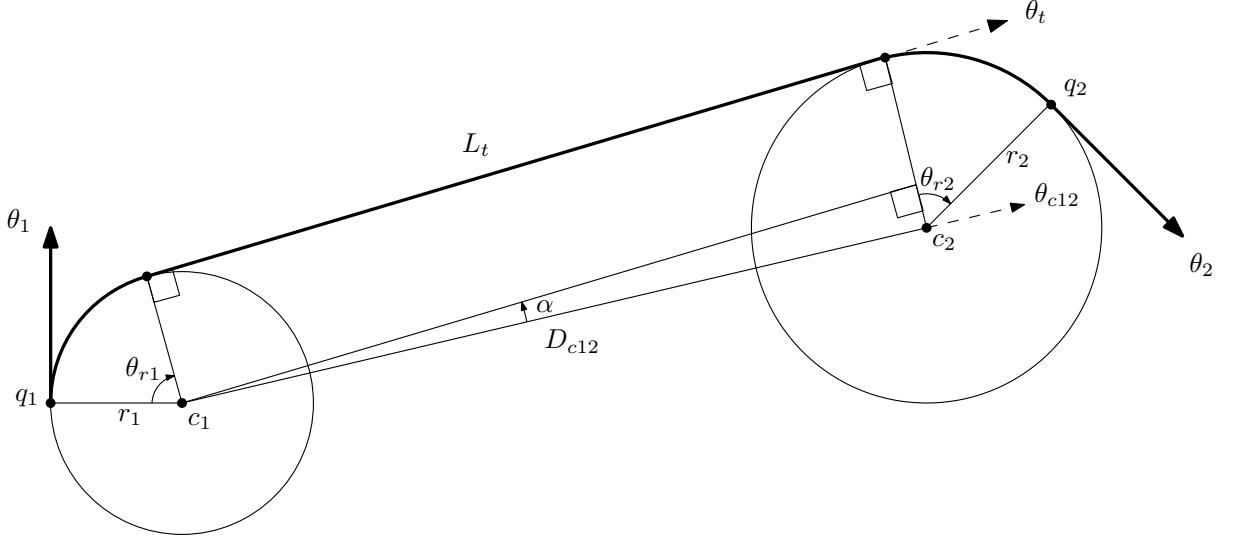


Figure A.2: RSR trajectory for different radii

The length of the trajectory can then be calculated as

$$L_{RSR} = \sqrt{D_{c12}^2 - (r_2 - r_1)^2} + r_1 \Delta_R(\theta_1, \theta_t) + r_2 \Delta_R(\theta_t, \theta_2). \quad (\text{A.1})$$

For the simplified case in which the circles are both equal to r_0 , we get the path length as

$$L_{RSR} = D_{c12} + r_0 (\Delta_R(\theta_1, \theta_t) + \Delta_R(\theta_t, \theta_2)).$$

Critical Value

There can be two kinds of critical values for an RSR trajectory. In the first kind, the type of trajectory after increasing the radius beyond the critical value is still an RSR trajectory while in the second kind, RSR trajectory does not exist.

When the radii become large, the value of the term $(\Delta_R(\theta_1, \theta_t) + \Delta_R(\theta_t, \theta_2))$ becomes greater than 2π .

$$(\Delta_R(\theta_1, \theta_t) + \Delta_R(\theta_t, \theta_2)) = \Delta_R(\theta_1, \theta_2) + 2\pi$$

This additional 2π causes the discontinuity. This is illustrated in figure A.3. At the critical value of r_1 , q_2 lies on circle 1. Thus, the first critical condition is that $(\Delta_R(\theta_1, \theta_t) + \Delta_R(\theta_t, \theta_2))$ should be less than 2π which is equivalent to

$$\Delta_R(\theta_1, \theta_2) > \Delta_R(\theta_1, \theta_t). \quad (\text{A.2})$$

For the second critical condition, keeping r_1 constant, the upper bound for r_2 is determined, beyond which the two circles do not have a common tangent. This occurs when one circle is completely inside another circle. At the critical value, the circles would be touching each other, such that one circle is inside the other circle. Let r_c denote this critical value.

$$\begin{aligned} \therefore d(c_1, c_2) &= |r_1 - r_2| \\ \therefore (y_1 - y_2 - r_1 \cos \theta_1 + r_2 \cos \theta_2)^2 + (x_1 - x_2 + r_1 \sin \theta_1 - r_2 \sin \theta_2)^2 &= (r_1 - r_2)^2. \end{aligned}$$

On expanding the above equation, we get

$$r_c = -\frac{(x_1 - x_2)^2 + (y_1 - y_2)^2 + 2r_1 \sin(t_1)(x_1 - x_2) - 2r_1 \cos(t_1)(y_1 - y_2)}{2(r_1 + \cos t_2(y_1 - y_2) - \sin t_2(x_1 - x_2) - \cos(t_1 - t_2))}. \quad (\text{A.3})$$

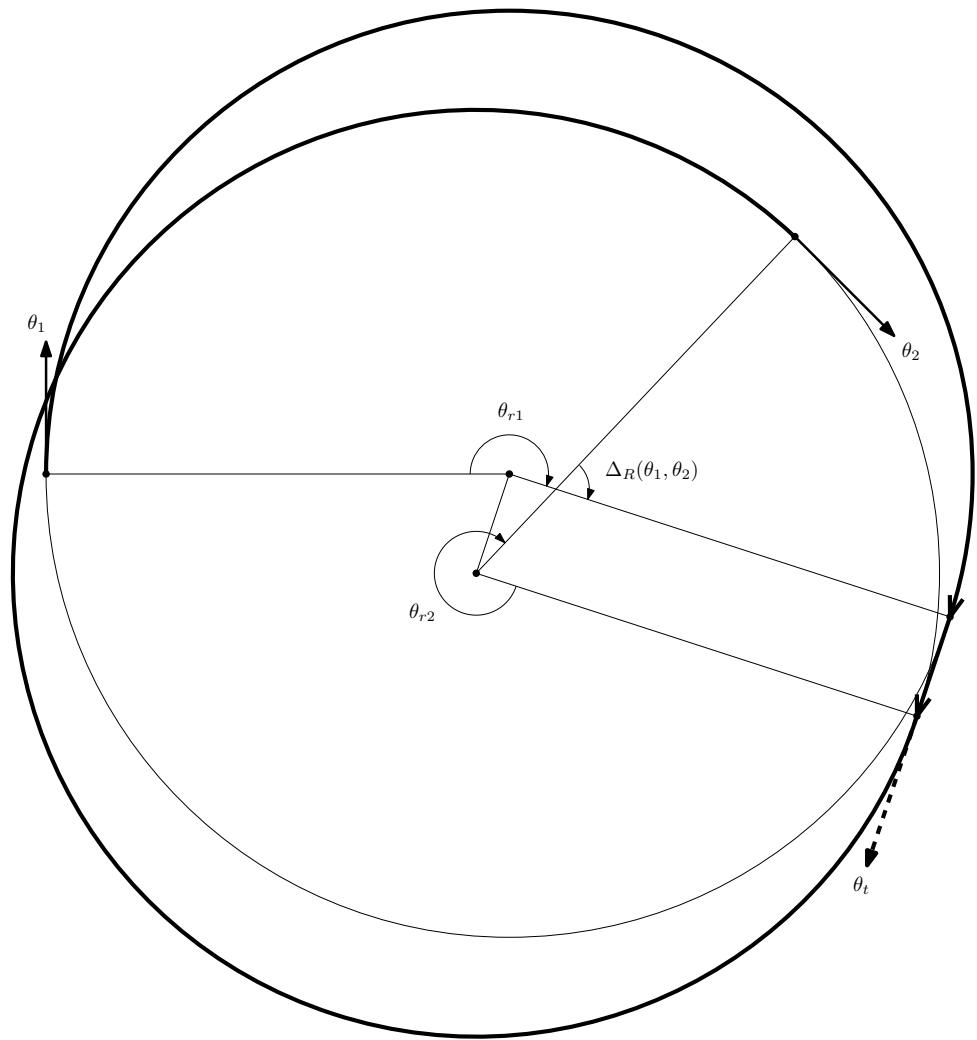


Figure A.3: RSR trajectory for high r_0

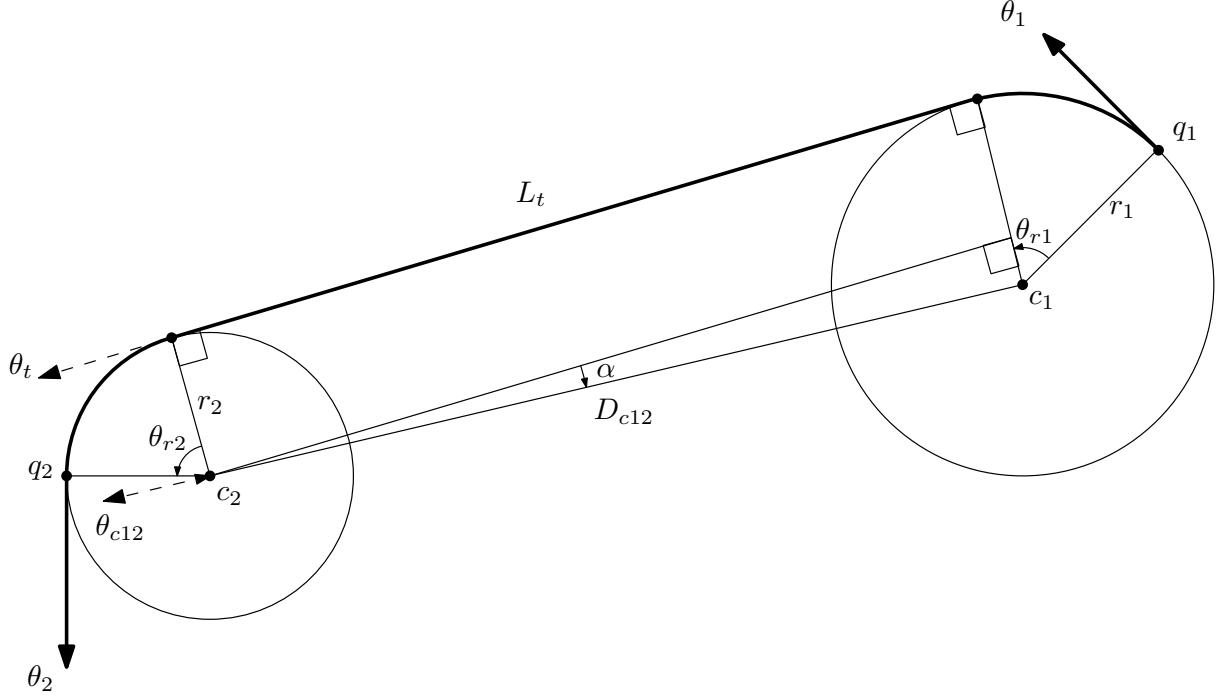


Figure A.4: LSL trajectory

The value of this expression can be negative. A negative value indicates that the circles always have a common tangent and hence, the upper bound does not exist.

A.2.2 LSL

The LSL trajectory is equivalent to a reverse RSR trajectory. Figure A.4 shows an LSL trajectory. The differences are in calculation of θ_t and L_{LSL} .

$$\begin{aligned}
 L_t &= \sqrt{D_{c12}^2 - (r_2 - r_1)^2} \\
 \therefore \alpha &= \arccos \left(\frac{\sqrt{D_{c12}^2 - (r_1 - r_2)^2}}{D_{c12}} \right) \\
 \theta_t &= \begin{cases} \theta_{c12} + \alpha & r_1 > r_2, \\ \theta_{c12} - \alpha & r_1 < r_2, \end{cases}
 \end{aligned}$$

$$L_{LSL} = \sqrt{D_{c12}^2 - (r_2 - r_1)^2} + r_1 \Delta_L(\theta_1, \theta_t) + r_2 \Delta_L(\theta_t, \theta_2). \quad (\text{A.4})$$

Critical Values

The first critical condition for an LSL path at which the trajectory type remains same is

$$\Delta_L(\theta_1, \theta_2) > \Delta_L(\theta_1, \theta_t). \quad (\text{A.5})$$

With r_1 constant, the upper bound on r_2 such that an LSL trajectory exists is

$$r_c = -\frac{(x_1 - x_2)^2 + (y_1 - y_2)^2 - 2r_1 \sin(t_1)(x_1 - x_2) + 2r_1 \cos(t_1)(y_1 - y_2)}{2(r_1 - \cos t_2(y_1 - y_2) + \sin t_2(x_1 - x_2) - \cos(t_1 - t_2))}. \quad (\text{A.6})$$

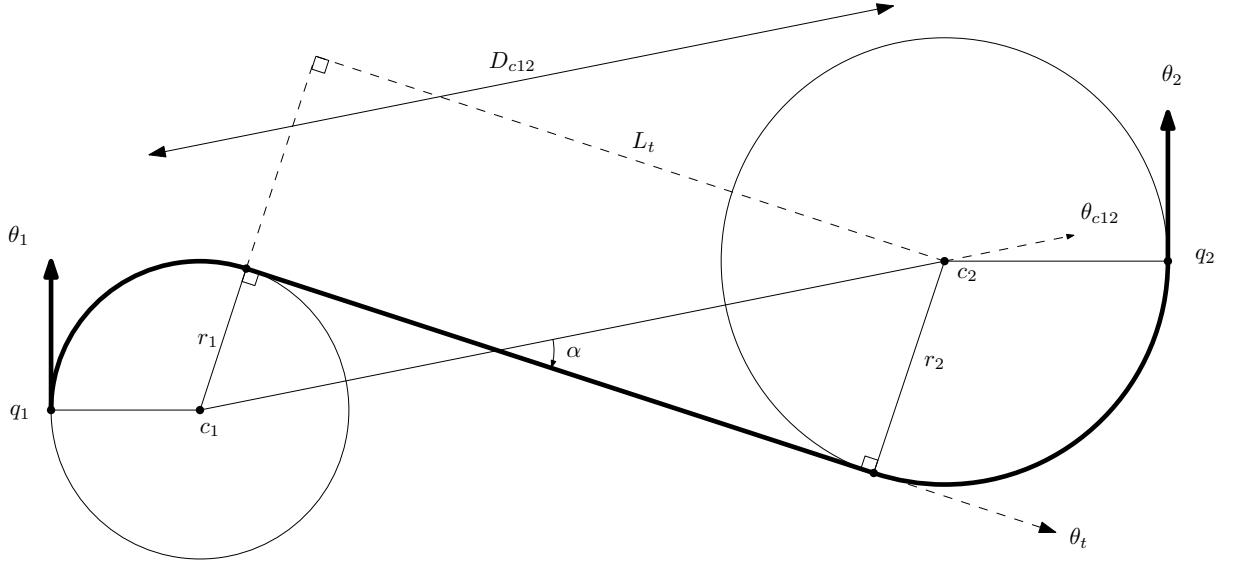


Figure A.5: RSL trajectory

A negative value of r_c implies that there is no upper bound.

A.2.3 RSL

An RSL trajectory can exist only when $D_{c12} > r_1 + r_2$. The tangent is given by

$$L_t = \sqrt{D_{c12}^2 - (r_1 + r_2)^2},$$

$$\theta_t = \theta_{c12} - \alpha,$$

where

$$\alpha = \arcsin \left(\frac{r_1 + r_2}{D_{c12}} \right).$$

The path length is given by

$$L_{RSL} = \sqrt{D_{c12}^2 - (r_1 + r_2)^2} + r_1 \Delta_R(\theta_1, \theta_t) + r_2 \Delta_L(\theta_t, \theta_2). \quad (\text{A.7})$$

Critical Value

Unlike RSR or LSL trajectories, RSL has only the second kind of critical value after which an RSL trajectory cannot exist. At this critical value, the two circles will touch each other externally.

$$\therefore d(c_1, c_2) = r_1 + r_2$$

With r_1 constant, the upper bound on r_2 is given by

$$r_c = \frac{(x_1 - x_2)^2 + (y_1 - y_2)^2 + 2r_1 \sin(t_1)(x_1 - x_2) + 2r_1 \cos(t_1)(y_1 - y_2)}{2(r_1 + \cos t_2(y_1 - y_2) - \sin t_2(x_1 - x_2) + \cos(t_1 - t_2))}. \quad (\text{A.8})$$

A negative value of r_c implies that there is no upper bound.

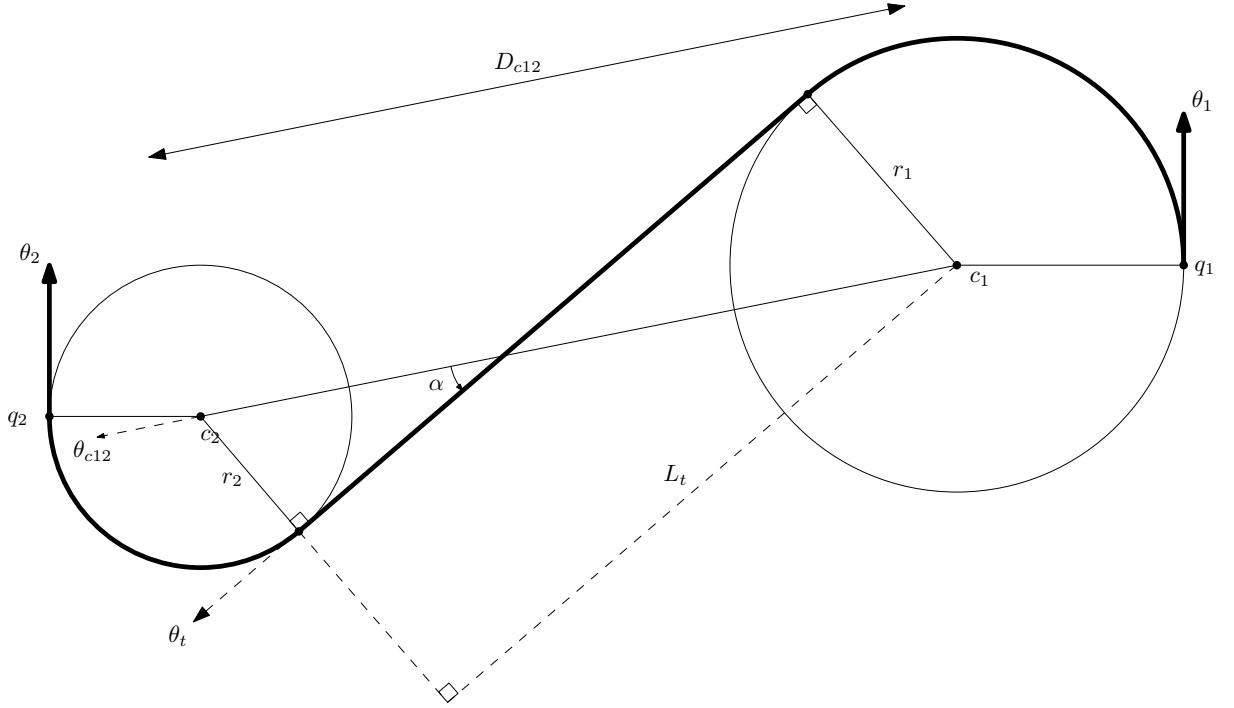


Figure A.6: LSR trajectory

A.2.4 LSR

An LSR path has been shown in figure A.6. The equations for LSR path are as follows:

$$\begin{aligned}
 L_t &= \sqrt{D_{c12}^2 - (r_1 + r_2)^2} \\
 \alpha &= \arcsin\left(\frac{r_1 + r_2}{D_{c12}}\right) \\
 \theta_t &= \theta_{c12} + \alpha \\
 \therefore L_{LSR} &= \sqrt{D_{c12}^2 - (r_1 + r_2)^2} + r_1 \Delta_L(\theta_1, \theta_t) + r_2 \Delta_R(\theta_t, \theta_2). \tag{A.9}
 \end{aligned}$$

Critical Value

With r_1 constant, the upper bound of the range of r_2 for which an LSR trajectory exists is given by

$$r_c = \frac{(x_1 - x_2)^2 + (y_1 - y_2)^2 - 2r_1 \sin(t_1)(x_1 - x_2) - 2r_1 \cos(t_1)(y_1 - y_2)}{2(r_1 - \cos t_2(y_1 - y_2) + \sin t_2(x_1 - x_2) + \cos(t_1 - t_2))}. \tag{A.10}$$

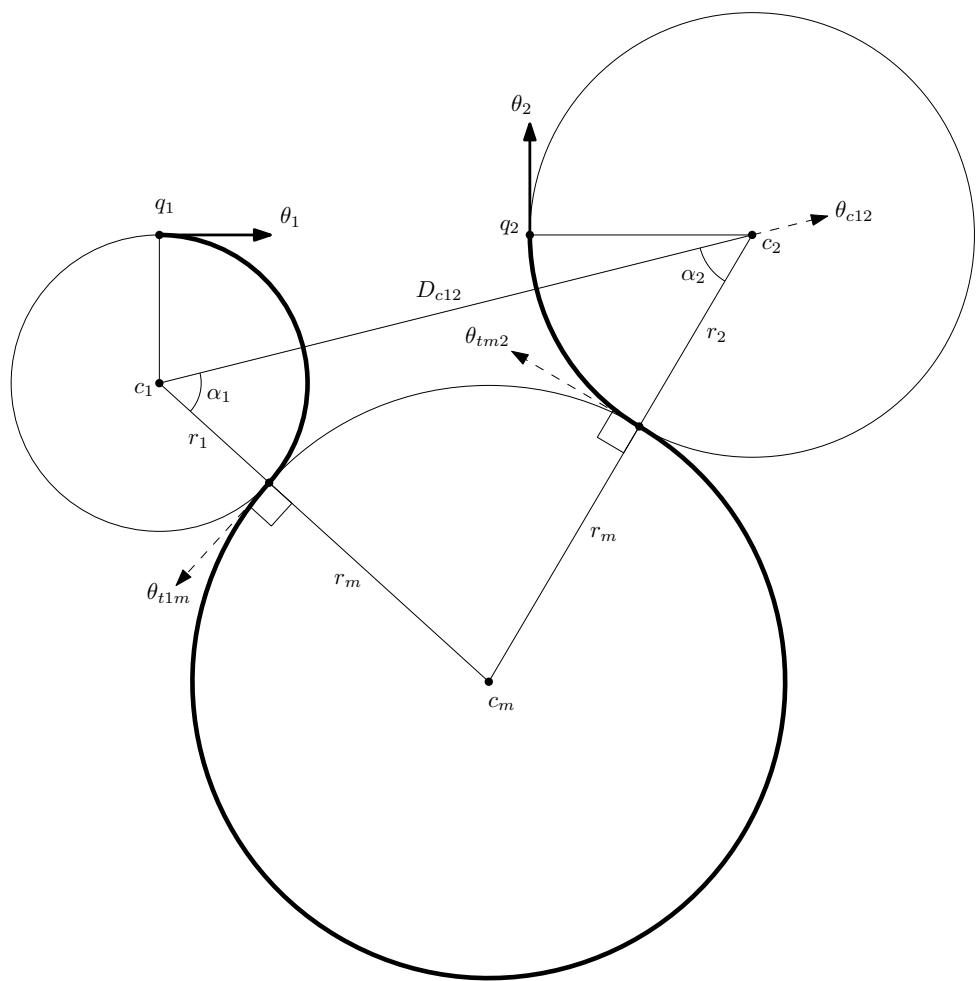


Figure A.7: RLR trajectory

A.2.5 RLR

Let the radius of the middle circle be r_m . The lengths of the sides of the triangle $c_1c_mc_2$ are known. Applying the law of cosines, we get

$$\begin{aligned}\alpha_1 &= \arccos\left(\frac{D_{c12}^2 + (r_1 + r_m)^2 - (r_2 + r_m)^2}{D_{c12}(r_1 + r_m)}\right), \\ \alpha_2 &= \arccos\left(\frac{D_{c12}^2 + (r_2 + r_m)^2 - (r_1 + r_m)^2}{D_{c12}(r_2 + r_m)}\right).\end{aligned}$$

Now,

$$\begin{aligned}\theta_{t1} &= \theta_{c12} - \alpha_1 - \pi/2 \\ \theta_{t2} &= \theta_{c12} + \alpha_2 + \pi/2.\end{aligned}$$

Therefore, the length is given by

$$L_{RLR} = r_1 \Delta_R(\theta_1, \theta_{t1}) + r_m \Delta_L(\theta_{t1}, \theta_{t2}) + r_2 \Delta_R(\theta_{t2}, \theta_2). \quad (\text{A.11})$$

A.2.6 LRL

An LRL trajectory shown in figure A.8 is equivalent to a reverse RLR trajectory. The equations are as follows

$$\begin{aligned}\alpha_1 &= \arccos\left(\frac{D_{c12}^2 + (r_1 + r_m)^2 - (r_2 + r_m)^2}{D_{c12}(r_1 + r_m)}\right), \\ \alpha_2 &= \arccos\left(\frac{D_{c12}^2 + (r_2 + r_m)^2 - (r_1 + r_m)^2}{D_{c12}(r_2 + r_m)}\right), \\ \theta_{t1} &= \theta_{c12} + \alpha_1 + \pi/2, \\ \theta_{t2} &= \theta_{c12} - \alpha_2 - \pi/2,\end{aligned}$$

$$L_{LRL} = r_1 \Delta_L(\theta_1, \theta_{t1}) + r_m \Delta_R(\theta_{t1}, \theta_{t2}) + r_2 \Delta_L(\theta_{t2}, \theta_2). \quad (\text{A.12})$$

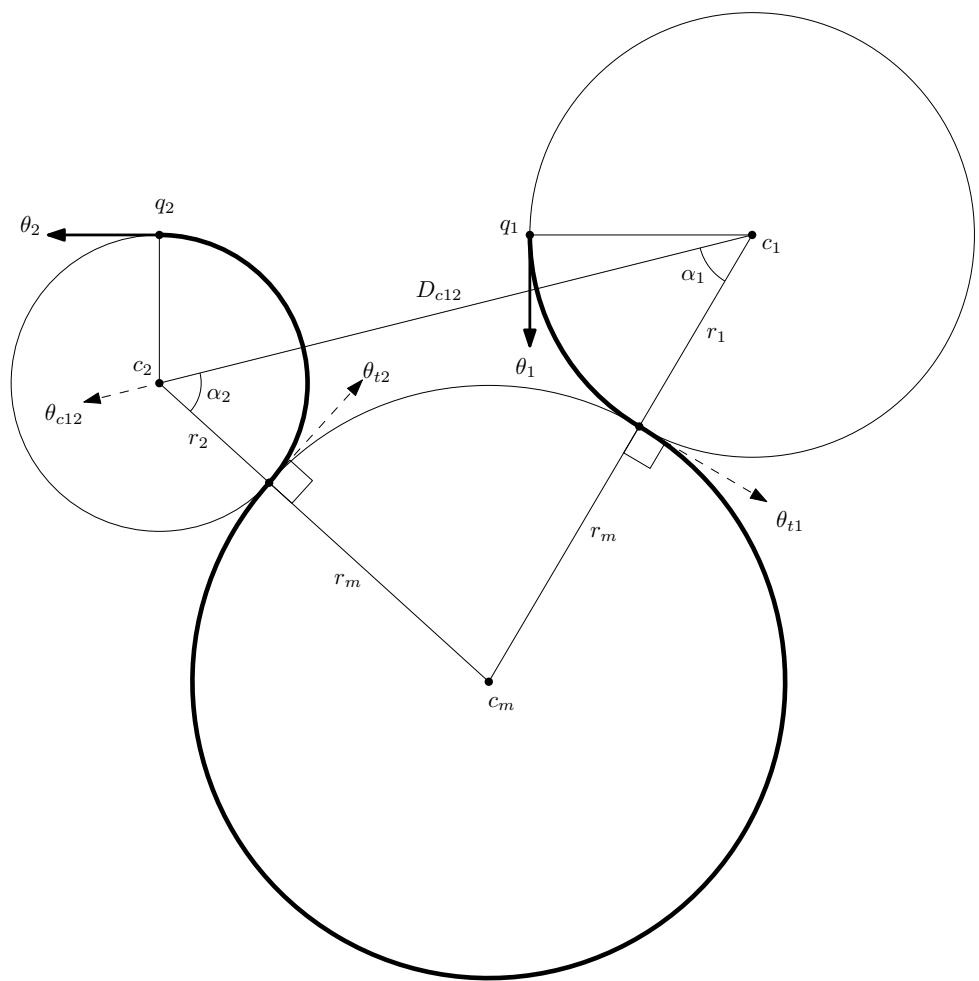


Figure A.8: LRL trajectory

Appendix B

Admissible Heuristics for A* search

The required conditions for a heuristic $h(x)$ to be admissible are as follows:

- $h(x_d) = 0$, where x_d is the destination point
- $h(x)$ is always less than the actual cost of reaching destination from x

Figure B.1 shows examples of admissible heuristics.

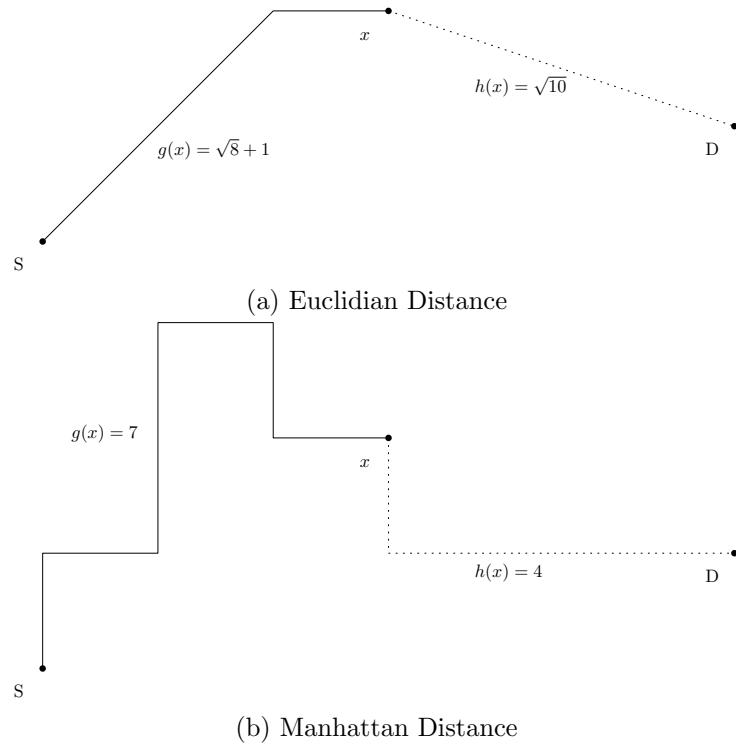


Figure B.1: Admissible Heuristics for length of path as the cost function

References

- [1] Wang, T., Wang, B., Wei, H., Cao, Y., Wang, M., and Shao, Z. Staying-alive and energy-efficient path planning for mobile robots. In *American Control Conference, 2008*, pages 868–873, 2008.
- [2] Tokekar, P., Karnad, N., and Isler, V. Energy-optimal velocity profiles for car-like robots. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1457–1462, 2011.
- [3] Liu, S. and Sun, D. Minimizing energy consumption of wheeled mobile robots via optimal motion planning. *Mechatronics, IEEE/ASME Transactions on*, PP(99):1–11, 2013. ISSN 1083-4435.
- [4] Product datasheets from nex robotics. <http://www.nex-robotics.com/>. Accessed: 14 July, 2014.
- [5] Product datasheets from mobile robots. <http://www.mobilerobots.com/ResearchRobots.aspx>. Accessed: 14 July, 2014.
- [6] Sun, Z. and Reif, J. On energy-minimizing paths on terrains for a mobile robot. In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, volume 3, pages 3782–3788 vol.3, 2003.
- [7] Ooi, C. C. and Schindelhauer, C. Minimal energy path planning for wireless robots. *Mob. Netw. Appl.*, 14(3):309–321, June 2009. ISSN 1383-469X.
- [8] Mei, Y., Lu, Y. H., Lee, C. S. G., and Hu, Y. C. Energy-efficient mobile robot exploration. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 505–511, 2006.
- [9] Plonski, P. A., Tokekar, P., and Isler, V. Energy-efficient path planning for solar-powered mobile robots. *Journal of Field Robotics*, 2013.
- [10] Barili, A., Ceresa, M., and Parisi, C. Energy-saving motion control for an autonomous mobile robot. In *Industrial Electronics, 1995. ISIE '95., Proceedings of the IEEE International Symposium on*, volume 2, pages 674–676, 1995.
- [11] Chitsaz, H., LaValle, S. M., Balkcom, D. J., and Mason, M. T. Minimum wheel-rotation paths for differential-drive mobile robots. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1616–1623, 2006.
- [12] Morales, J., Martinez, J. L., Mandow, A., Garcia-Cerezo, A. J., and Pedraza, S. Power consumption modeling of skid-steer tracked mobile robots on rigid terrain. *Robotics, IEEE Transactions on*, 25(5):1098–1108, 2009. ISSN 1552-3098.

- [13] Mei, Y., Lu, Y. H., Hu, Y. C., Hu, and Lee, C. S. G. Energy-efficient motion planning for mobile robots. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 5, pages 4344–4349. IEEE, 2004.
- [14] Dubins, L. E. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79(3):pp. 497–516, 1957. ISSN 00029327.
- [15] Hart, P. E., Nilsson, N. J., and Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968. ISSN 0536-1567.
- [16] Nash, A., Daniel, K., Koenig, S., and Felner, A. Theta*: Any-angle path planning on grids. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, pages 1177–1183. AAAI Press, 2007. ISBN 978-1-57735-323-2.