Obstacle avoidance using optimal control theory

H.L.Hagenaars
Report No. DCT 2004.36

Report of traineeship at Tokyo Institute of Technology

Coach: Dr. J. Imura (Tokyo Institute of Technology)
Supervisor: Prof. dr. H. Nijmeijer (TU/e)
Abstract

This report addresses the optimal control problem with non-convex state constraints. The specific application pursued is the obstacle avoidance problem. To solve the problem, discretization of time and space is used, which is often used in solving obstacle avoidance problems. During each time step, the optimal control using the standard continuous-time cost function is derived in an explicit form, including the optimal next step as a parameter. The optimal next step can be obtained by solving a discrete minimization problem. An efficient algorithm to solve this problem is presented. Numerical simulations are used to point out that the standard continuous-time cost function does not lead to satisfactory results. Next, a new cost function is proposed that includes the intermediate desired target states. This leads to a new approach that optimizes the continuous-time behavior as well as the discrete states simultaneously. The optimal control can be given in an explicit form that includes the intermediate target states as parameters, and the optimal intermediate target states can be obtained by solving a discrete optimization problem, making use of the algorithm presented before. This approach gives a sub-optimal solution to the initial problem. Finally, numerical simulations illustrate the effectiveness of the new approach.
## Contents

Introduction 1

1 Problem statement 3  
1.1 Problem statement 3  
1.2 Problem solution 5

2 Numerical search algorithms 7  
2.1 Graph theory and shortest path algorithms 7  
2.2 Forward Dynamic Programming 8  
2.3 Breadth First Search 9

3 Numerical simulation results 13  
3.1 A simple example 13  
3.2 Evaluation of the results 15

4 Restatement of the problem 17  
4.1 Proposal of a new cost function 17  
4.2 Derivation of optimal controller 18  
4.3 Calculation of the optimal intermediate target states 20

5 Numerical simulations of the final result 21  
5.1 A simple example 21  
5.2 Performance of the used algorithm 23

6 Conclusions and recommendations 25  
6.1 Conclusions 25  
6.2 Recommendations 26

Bibliography 27

A Calculation of $W$ 29

B Breadth First Search Algorithm 31

C Alternative numerical search algorithms 39
Introduction

The problem of planning an optimal trajectory of a mobile robot avoiding (possibly moving) obstacles is one of the challenging topics in the field of mobile robotics, since it belongs to the class of optimal control problems of continuous-time dynamical systems under a nonconvex constraint set and hence it is very difficult to rigorously solve it.

A well-known approach is to add several potential functions for avoiding the obstacles to the original cost function; in that case, the obstacle avoidance problem is described as the optimal control problem without constraints. Since the optimal control problem in this case includes no constraints, the gradient method is mainly used, which is suitable for real-time optimization. However, the obtained minimum often is a local minimum, and the optimization algorithm may fall into some deadlock even if an optimal solution exists. So how to properly construct the potential functions is one of the significant issues, that makes it difficult to solve such a problem in an adequate way.

One realistic approach is to use a discretization of time and state space to formalize the obstacle avoidance problem in terms of the discrete optimization problem. This approach makes it possible to apply efficient numerical search algorithms such as the Dijkstra method and the A* method. However, since the minimization of the discrete-time cost function on the discrete set of the state is mostly considered in this approach, a continuous-time trajectory that interpolates between the obtained optimal discrete states will have to be calculated for the implementation. In that case only an approximate optimization will be performed. A disadvantage of such an approach is that the discrete states and the continuous-time behavior will not be optimized simultaneously.

In this report a discretization of time and state space is also used. However, an approach is proposed that optimizes the continuous-time behavior as well as the discrete states simultaneously in the obstacle avoidance problem. In the first chapter, the problem will be stated with a standard optimal control cost function. An optimal controller that solves this problem will be derived. The next optimal discrete state appears in the found controller as
a parameter. This optimal next discrete state can be obtained solving a
discrete minimization problem. In the second chapter, an algorithm that
efficiently solves the minimization problem will be presented and the com-
putational cost required to solve this problem will be discussed. Numerical
simulations are used to illustrate that the used cost function does not lead to
satisfactory results on the non-convex constraints set. Therefore, in chapter
4, a new cost function will be proposed that includes the intermediate de-
sired states. This new cost function leads to a restatement of the problem.
Next, the continuous-time input and the intermediate target states minimiz-
ing this cost function are derived. The optimal input is given in an explicit
form including the intermediate target states as parameters, and the optimal
intermediate target states are obtained by solving a discrete optimization
problem for which an efficient algorithm already has been presented. The
obtained solution to the discretized problem is a sub-optimal solution to the
initial problem. In chapter 5, numerical simulation shows the effectiveness of
the proposed approach and of the used search algorithm. Finally conclusions
are drawn and some recommendations for future research are given.
Chapter 1

Problem statement

The goal of the research presented in this report is to solve the obstacle avoidance problem using optimal control theory. The special application pursued in this report is the case of a mobile robot moving through a field with obstacles. In this chapter, the problem statement will be given and an optimal controller will be derived for the problem.

1.1 Problem statement

Fig. 1.1 gives a simple two-dimensional case of an obstacle avoidance problem. The intended goal in this report is to obtain a piecewise continuous control $u(t)$ that drives a mobile robot, starting in point $x_0$, to the final state $x_f$ using optimal control theory. As can be seen in Fig. 1.1, due to the obstacles present in the field, the state $x(t)$ of the dynamical system considered cannot take arbitrary positions. Therefore, the state $x(t)$ must be included in a constraint set $C(t)$, where $C(t)$ is some given time-invariant or time-varying,

Figure 1.1: A simple 2 dimensional obstacle avoidance problem
non-convex subset of $\mathbb{R}^n$. The non-convex constraint set is the key issue in this report. Under these contraints, the problem can be classified as a continuous time optimal control problem subject to a, possibly varying, non-convex constraint set. This problem can be formulated formally as follows:

**Problem 1.1** Given the initial and final states $(x_0, x_f)$, find $u(t) \in \mathcal{PC}^m$, where $\mathcal{PC}$ is the set of all piecewise continuous functions, minimizing

$$J(x_0, u) = \int_{t_0}^{t_f} \{ x^T(t)Qx(t) + u^T(t)Ru(t) \} \, dt$$

subject to:  
$$\dot{x} = Ax + Bu$$  
$$x \in \mathbb{R}^n, u \in \mathbb{R}^m$$  
$$x(t) \in C(t), \forall t \in [t_0, t_f), \quad x(t_f) = x_f$$

where $C(t)$ is a time-invariant or time-varying, non convex subset of $\mathbb{R}^n$, $\mathbb{Q} \geq 0$ and $R > 0$. Furthermore the pair $(A, B)$ is assumed to be controllable and the pair $(Q^{1/2}, A)$ is assumed to be observable.

Although the cost function stated in Problem 1.1 is known from the standard optimal control problem formulation [1], it is difficult to guarantee the that the strong constraint $x(t) \in C(t)$ in Problem 1.1 will hold at all times. Therefore, the non-convex constraints will be relaxed by discretizing the state space and time-axis. This well-known approach is widely used in solving obstacle avoidance problems [9, 11, 13]. Taking the discretization into account, the problem can be formulated as follows:

**Problem 1.2** Given the initial and final states $(x_0, x_f)$, find $u(t) \in \mathcal{PC}^m$, where $\mathcal{PC}$ is the set of all piecewise continuous functions, minimizing

$$J(x_0, u) = \sum_{i=0}^{f-1} \int_{t_i}^{t_{i+1}} \{ x^T(t)Qx(t) + u^T(t)Ru(t) \} \, dt$$

subject to:  
$$\dot{x} = Ax + Bu$$  
$$x(t_i) = x_i \in \mathcal{Cd}(t_i), \quad i = 0, 1, \ldots, f$$

where $t_i = ith$, $h$ denotes the discrete time interval, $\mathcal{Cd}(t_i)$ is a time-invariant or time-varying discrete set in $\mathbb{R}^n$ approximating $C(t)$, $\mathbb{Q} \geq 0$, $R > 0$, and the pair $(A, B)$ is assumed to be controllable and the pair $(Q^{1/2}, A)$ is assumed to be observable.

The discrete set $\mathcal{Cd}(t_i)$ will be obtained by discretizing the continuous set $C(t_i)$ in a suitable way, and the constraint condition will be satisfied only at
1. Problem statement

every discrete time $t_i$, i.e. $x(t_i) \in C_d(t_i)$. In this way, the discretization of time and space is used to relax the non convex constraints. Note that the solution to Problem 1.1 is sub-optimal for the original problem. Although the best way to discretize the state space for this specific problem might be an interesting topic of research, this problem will not be treated in this report.

1.2 Problem solution

A solution to Problem 1.2 is given as follows:

**Theorem 1.1** For Problem 1.2, the optimal control $u^* \in \mathcal{P}_C$ is given by:

$$u^*(t) = -R^{-1}BPx(t) - R^{-1}BR^T e^{AT(t_{i+1}-t)} W^{-1} (e^{Ah}x_i - x_{i+1})$$ (1.3)

where $P$ and $W$ are the real, positive-definite solutions of

$$PA + A^TP - PBR^{-1}B^TP + Q = 0$$ (1.4)
$$W \bar{A}^T + \bar{A}W + BR^{-1}B^T - e^{Ah}BR^{-1}B^T e^{Ah} = 0$$ (1.5)

respectively, and

$$\bar{A} = A - BR^{-1}B^TP$$ (1.6)

Furthermore, $x^*_i$, $i = 1, 2, \ldots, f-1$, are the solutions to the following discrete optimization problem.

$$\min_{x_i \in C_d(t_i), i=1,2,\ldots,f-1} \sum_{i=0}^{f-1} (e^{Ah}x_i - x_{i+1})^T W^{-1} (e^{Ah}x_i - x_{i+1})$$ (1.7)

**Proof** Letting $J_i(x_i, x_{i+1}, u_i) = \int_{t_i}^{t_{i+1}} [x^T(t)Qx(t) + u^T(t)Ru(t)] dt$, using (1.4), the following result can be obtained

$$J_i(x_i, x_{i+1}, u_i) = x^T(t_i)Px(t_i) - x^T(t_{i+1})Px(t_{i+1}))$$
$$+ \int_{t_i}^{t_{i+1}} \left\{ u_i(t) + R^{-1}B^TPx(t) \right\}^T R \left\{ u_i(t) + R^{-1}B^TPx(t) \right\} dt.$$ (1.8)

Let $u_i(t) = v_i(t) - R^{-1}B^TPx(t)$, then the problem is reduced to the minimum energy control problem with respect to $v_i$; for given $x_i$ and $x_{i+1}$, find a control $v_i^*$ satisfying $(x(t_i), x(t_{i+1})) = (x_i, x_{i+1})$ and minimizing the cost

$$\tilde{J} = \int_{t_i}^{t_{i+1}} v_i^T(t)Rv_i(t) dt,$$ in which the optimal controller and the minimum cost are given by

$$v_i^*(t) = -R^{-1}B^T e^{AT(t_{i+1}-t)} W^{-1} (e^{Ah}x_i - x_{i+1})$$
$$\min \tilde{J} = (e^{Ah}x_i - x_{i+1})^T W^{-1} (e^{Ah}x_i - x_{i+1}),$$
respectively [7, 6]. It is remarked here that $W$ is a positive-definite matrix given by $W = \int_0^t e^{A\tau}BR^{-1}B^Te^{A^T\tau}d\tau$, which is a solution to (1.5) (see Appendix A).

Theorem 1.1 gives the optimal input $u^*(t)$ in an explicit form. The optimal next step $x_{i+1}$ that gives the minimum value of the cost function (1.2), can be obtained by solving the discrete minimization problem stated in (1.7). There are many numerical algorithms available to solve this problem. What kind of algorithms are available and which one of these algorithms is most suitable in this approach will be discussed in the next chapter.
Chapter 2

Numerical search algorithms

In the previous chapter it was pointed out that, in order to find the next optimal step $x_{i+1}$, a discrete minimization problem, given in (1.7), has to be solved. Because the discretized state space can be represented as a graph whose vertices are points in $C_d(t_i)$, this problem is equivalent to finding the shortest path in a graph. In this chapter some fundamentals of graph theory will be explained and an algorithm that can solve this specific problem in an efficient way will be presented.

2.1 Graph theory and shortest path algorithms

A graph is a structure consisting of a set of nodes, also called vertices, and a number of edges, each of them linking some pair of vertices. Fig.2.1 shows a part of a graph that represents the problem to be solved stated in the previous section (suppose $x_i \in \mathbb{R}$). The vertices in the graph are rendered as black dots. The black crosses are nodes that cannot be visited because there is an obstacle present at that discrete point. The edges connecting the vertices are given by the solid black arrows. Distinction can be made between directed graphs and undirected graphs. In a directed graph, an edge goes from one vertex to the other. In an undirected graph however, the relation between connected vertices is symmetric [4, 5]. The arrowhead of each edge in Fig.2.1 makes clear that the graph in the figure is a directed graph. To each edge, a certain kind of weight can be adjudged. Another important property of the graph in Fig.2.1 is the absence of cycles. A cycle is a set of edges forming a loop, and all pointing the same way around the loop.

As stated before, the solution to the discrete minimization problem comes down to finding the shortest path in the graph whose vertices are the discrete points in $C_d$. In Fig.2.1, that could represent a part of the graph in this
2. Numerical search algorithms

Figure 2.1: Interpretation of the discrete optimization problem by a graph

2.1 Interpretation of the discrete optimization problem by a graph

specific problem, the graph has a staged structure caused by the discrete time series, obtained from discretizing time. At each stage, i.e. at each time-step, the state space \( C_d(t_i) \) consists of a finite set of discrete values. The constructed graph consists of a limited number of these stages and the goal is to find the 'cheapest' trajectory to get from the initial condition to the final state in the last stage. In this case 'cheapest' can be measured simply as the sum of the edge weights of a path. Considering (1.7), the edge weight between two vertices \( x_i \) and \( x_{i+1} \), \( K(x_i, x_{i+1}) \) can be calculated using:

\[
K(x_i, x_{i+1}) = (e^{Ah}x_i - x_{i+1})^2W^{-1}(e^{Ah}x_i - x_{i+1})
\]  

(2.1)

Note that the weight \( K \) of an edge is greater than or equal to 0 in this case, which implies that there are no negative weights in the graph. Many efficient numerical search algorithms are available to solve this kind of problem, e.g. Dijkstra's algorithm and the A* algorithm (see appendix C) [5]. The specific characteristics of the constructed graph in this case, can be exploited to solve the discrete minimization problem (1.7) in a more efficient way by using a specific class of search algorithms called forward dynamic programming.

2.2 Forward Dynamic Programming

Forward dynamic programming (FDP) algorithms can be used to find the shortest path in a graph. These algorithms however, require a specific dynamic structure of the graph. For FDP problems, the nodes in the graph
2. Numerical search algorithms

have to be organized into a sequence of stages. The state of the system has to be known at the beginning of the process, so the first stage has only one node, the initial condition. Subsequent stages may have several possible states, each depicted by a different node. In Fig.2.1 for example, the nodes for a given stage are arranged in a vertical column that represents different 'states' of the system at that stage, and the collection of these states determines the state space \( x_i \in \mathcal{C}_d(t_i) \) for that stage. The edges in the graph have to be directed, and they have to represent 'transitions' from a state in stage \( k \) to a state in stage \( k + 1 \). There is a weight adjudged to each of these transitions and the problem is to find the path through the graph from beginning to end for which the total of the transition values is minimized. In this context, the procedure of minimizing one step at a time does not work, since using this approach in the first few stages may lead to a state from which only very high weighted edges are available. In FDP, the early decisions are made with the later stages being taken into consideration, so that the total sum of all transitions is optimized.

The graph constructed from the discrete set \( \mathcal{C}_d \) satisfies all the criteria mentioned above, so a FDP algorithm can be used to solve the discrete minimization problem efficiently. Consider (1.7), the FDP approach for the problem can be expressed in a mathematical way. Defining

\[
V_k(x_k) := \min_{x_{i-1}, \ldots, x_{k-1}} \sum_{i=0}^{k} f(x_{i-1}, x_i),
\]

the following relation can be obtained, that expresses the minimal costs to reach stage \( k + 1 \)

\[
V_{k+1}(x_{k+1}) = \min_{x_k} \{ f(x_k, x_{k+1}) + V_k(x_k) \}.
\]

From the description above it is clear that FDP can be used to find the optimal next step \( x_{i+1} \) that appears in the optimal control \( u^* \) in (1.3) as a parameter. An efficient FDP algorithm is the Breadth First Search algorithm.

2.3 Breadth First Search

The Breadth First Search (BFS) algorithm, is a search algorithm in which the nodes are searched stage by stage (opposite to e.g. the depth first search algorithm that explores one branch to the last stage at a time). The BFS algorithm is typically implemented using a queue. From the starting vertex (the initial condition) the adjacent vertices are explored and saved in the queue, if that vertex has not already been queued before. After exploring a vertex, the next vertex in the queue will be explored. The algorithm
2. Numerical search algorithms

Figure 2.2: A part of the graph for $n_{out} = 3$ and $n_v = 2$

terminates when the queue is empty. At that time all the vertices have been explored. An implementation of the used BFS algorithm in this research can be found in Appendix B.

The computational cost of the BFS algorithm depends on the number of edges in the graph that have to be searched. If the total number of the edges in the graph is $n_e$, the computational cost of the BFS algorithm is given by $O(n_e)$.

The number of edges in the graph, and therefore the computational cost to search the graph, is strongly dependent on several parameters. Consider for example a mobile robot, which can only move in one direction. So the state of the system is given by the position $x_p \in \mathbb{R}$ and its velocity $x_v \in \mathbb{R}$. Suppose that the position region is restricted by several obstacles and the velocity is not restricted. Fig.2.2 shows a part of the graph that could represents the discretized space in this case. The discretization of the velocity consists of only two discrete values. The gray discrete position cannot be visited because of the presence of an obstacle in that point. Let $f$ denote the number of prediction steps, i.e. the number of stages in the graph, $n_v$ denote the number of discrete velocities at each $x_p$, $n_n$ denote the total number of the nodes in the graph including the discrete positions as well as the discrete velocities, also the nodes in which the obstacles are present, for all prediction steps, $n_{out}$ denote the maximum number of edges outgoing from each node, and $n_e$ denote the number of edges in the graph
2. Numerical search algorithms

including the discrete positions and velocities (not including the nodes in which the obstacles exist). For example, in Fig.2.2, for each position \( x_p \), the velocity can take to discrete values, so in this case \( n_v = 2 \). Each position \( x_p \) is connected to three other positions and therefore \( n_{out} = 3 \). In the general case, the total number of nodes is given by

\[
n_k = n_v \sum_{k=0}^{f} \{k(n_{out} - 1) + 1\} = n_v \frac{1}{2}(n_{out} - 1)f^2 + \frac{1}{2}(n_{out} + 1)f + 1
\]

Thus the total number \( n'_e \) of edges in the graph including the discrete positions as well as the discrete velocities and the nodes in which the obstacles exist, equals

\[
n'_e = n^2_v n_{out} \sum_{k=0}^{f-1} \{k(n_{out} - 1) + 1\}. \tag{2.2}
\]

Furthermore, for each obstacle, \( 2n^2_v n_{out} \) edges in general will be lost. In Fig.2.2, the edges lost because of the presence of the obstacle are represented by the dashed lines going towards and leaving the obstacle. Let \( P \in [0,1] \) denote the uniform probability density for emergence of obstacles in the prediction area. Then if \( P < 0.5 - \varepsilon \), the number of edges is approximately given by

\[
n_e \approx n'_e - 2Pn^2_v n_{out} \sum_{k=0}^{f-1} \{k(n_{out} - 1) + 1\} - Pn^2_v n_{out} \{f(n_{out} - 1) + 1\} = (1 - 2P)n'_e - Pn^2_v n_{out} \{f(n_{out} - 1) + 1\}. \tag{2.2}
\]

The last term in this equation represents the number of edges lost, due to the obstacles present in the last prediction step. As can be seen in (2.2) the number of edges decreases proportionally for increasing probability density of the obstacles \( P \). Consider the case \( f = 10, n_v=1 \) and \( n_{out} = 3 \). If there are no obstacles present, the number of edges \( n_e \) will be 300. However, if the uniform probability density of the obstacles \( P = 0.2 \), the number of edges decreases to \( n_e = 195 \).

Note also that in the above approach the discretization applies to the whole state space \( C(t) \), not only to a part of the state space for example the position of the system. Indeed, it may be useful that the discretization of only a part of the state space can be applied. However in this case, the above BFS algorithm cannot directly be applied. Such an extension will not be treated in this report.
2. Numerical search algorithms
Chapter 3

Numerical simulation results

In the first two chapters a control strategy has been derived and a method to find the optimal next step $x_{i+1}$ that appears in the optimal controller as a parameter, has been presented. In this chapter, the effectiveness of the obtained control strategy will be tested using numerical simulations.

3.1 A simple example

Consider the simple vehicle in Fig.3.1. If the mass of the vehicle $m = 1$, the dynamics of the vehicle are given by:

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u,$$

where $x = \begin{bmatrix} x_p \\ x_v \end{bmatrix}$,

where $x_p$ is the position of the vehicle motion restricted to one direction, $x_v$ is its velocity and $u$ is the input force acting on the system.

Suppose that the system is moving through a field with obstacles at constant velocity in the direction perpendicular to $x_p$. The discretized region of $x_p$.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure3.1.png}
\caption{A simple cart moving in one dimension}
\end{figure}
3. Numerical simulation results

Figure 3.2: Results

consists of a set of 10 discrete points at each time $t_i$, and the distribution of the obstacles in the field is random. The discrete time interval $h = 1 \text{s}$. The discretization of the velocity $x_v$ consists of only 1 velocity, $x_v = 0$. For simplicity the maximum motion of the system is restricted to one grid point at each time-step, that is, the number of edges from a node to the nodes at the next time-step is less than or equal to 3. A receding horizon policy is used for control of the vehicle, that is, at time-step $t_i$ the optimal control input $u^*(t)$ is computed for 20 prediction steps, i.e., $[t_i, t_{i+20}]$, and $u^*(t)$ is applied only for $t \in [t_i, t_{i+1})$. Furthermore, the final state $x_f$ will not be fixed in each optimal control problem here, $x_f$ is also optimized in the same way as $x_i$, $i = 1, 2, \ldots, f - 1$, in (1.3). The weighting matrices of the cost function (1.2) are given by

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad R = 1$$

Fig.3.2 shows the results of the first 40 steps. It is obvious that the obtained optimal route through the obstacle field is not the shortest route possible. The 'cheapest' route trough the obstacles obtained by the BFS algorithm, is the route that minimizes the distance to the origin $x_p = 0$ each step. Fig.3.3 shows a zoomin part of Fig.3.2. This figure illustrates that the continuous input in between each time step also tries to stabilize the system at origin.
3. Numerical simulation results

3.2 Evaluation of the results

Although the obtained result is definitely not the optimal way to solve the problem, it is the result that could have been expected from the used cost function (1.2). The minimum value of the cost function is reached when \( x = 0 \). Finding a continuous input to drive a system from initial state \( x_0 \) to final state \( x_f \), minimizing this cost function, leads to an input that tries to stabilize the system at origin each time step. To stabilize the system in the final desired state \( x_f \), the cost function can be modified to:

\[
J(x_0, u) = \sum_{i=0}^{f-1} \int_{t_{i+1}}^{t_i} \left\{ (x(t) - x_f)^T Q(x(t) - x_f) + u^T(t) R u(t) \right\} dt \tag{3.1}
\]

Fig. 3.4 however, illustrates that also this cost function does not lead to satisfactory results. The solid line in Fig.3.4 gives the optimal trajectory that will be obtained when using the cost function given in (3.1). While driving the system from \( x_0 \) to \( x_f \), the controller will minimize the distance to the final target state \( x_f \) during each time-step. Again this is not the optimal trajectory, that is, it is not the shortest path through the field with obstacles, a trajectory that one would intuitively consider optimal.

The solid line in Fig.3.5 however, indicates the path that is optimal in this case. Thus it will be required to take another, more suited cost function to produce an outcome closer to what is 'expected' or desired. One way to do so is to include the intermediate desired target states \( x_f', x_f' \) and \( x_f'' \) (see Fig.3.5) to be addressed into the cost function. Such a change in the cost
3. Numerical simulation results

Figure 3.4: Undesirable trajectory when including $x_f$ in the cost function

Figure 3.5: Desired trajectory and the intermediate states

function gives the ability to optimize the intermediate target states and the trajectory in the time interval simultaneously.
Chapter 4

Restatement of the problem

The previous chapter has clarified that the used cost function does not lead to desired results. To obtain more desirable results, the intermediate target states have to be included in the cost function. In this chapter a new cost function will be proposed which leads to a new problem formulation. Also, the solution to this new problem will be derived.

4.1 Proposal of a new cost function

When the intermediate desired target states are included in the cost function the problem can be restated as follows.

**Problem 4.1** Given the initial and final states \((x_0, x_f)\). Find \(u(t) \in PC^m\) and the intermediate target states \(x_i \in \mathbb{R}^n, i = 1, 2, \ldots, f - 1\), minimizing

\[
J(x_0, u, \{x_i\}_{i=1,2,\ldots,f}) = \sum_{i=0}^{f-1} \int_{t_i}^{t_{i+1}} \{(x(t) - x_{i+1})^T Q(x(t) - x_{i+1}) + u^T(t) R u(t)\} dt
\]

subject to: \[\dot{x} = Ax + Bu\]
\[x(t_i) = x_i \in C_d(t_i), \quad i = 0, 1, \ldots, f\]

where \(t_i = ih\), \(h\) denotes the discrete time interval, \(C_d(t_i)\) is a time-invariant or time-varying discrete set in \(\mathbb{R}^n\) approximating \(C(t)\), \(Q \geq 0, R > 0\), and the pair \((A, B)\) is controllable and the pair \((Q^{1/2}, A)\) is assumed to be observable.

In this problem, the intermediate target states \(x_i, i = 1, 2, \ldots, f - 1\), are introduced in the cost function (4.1). The intermediate desired target state
will be optimized together with the input so as to minimize the cost function, which makes it possible to derive the desired trajectory as described at the end of the previous chapter.

4.2 Derivation of optimal controller

A solution of Problem 4.1 is given as follows.

**Theorem 4.1** For Problem 4.1, an optimal input $u^* \in \mathcal{P}C$ is given by

$$u^*(t) = -R^{-1}BP(x(t) - x^*_{i+1}) + R^{-1}B^T \bar{A}^TPA x^*_{i+1}$$

$$-R^{-1}B^T e^{\bar{A}(t_{i-1} - t)} W^{-1} (Dx_i^* - E_{i+1}^*)$$

where $P$ and $W$ are the real, positive-definite solutions of

$$PA + \bar{A}^TP - PB R^{-1}B^TP + Q = 0$$

$$W\bar{A}^T + \bar{A}W + BR^{-1}B^T - e^{\bar{A}h}BR^{-1}B^T e^{\bar{A}T h} = 0$$

respectively, and

$$\bar{A} = A - BR^{-1}B^TP, \quad D = e^{\bar{A}h},$$

$$E = \int_0^h e^{\bar{A} \tau} d\tau (I + BR^{-1}B^T \bar{A}^T P)A - e^{\bar{A}h},$$

and $x^*_i, i = 1, 2, \ldots, f - 1,$ are the solutions to the following discrete optimization problem.

$$\min_{x_i \in \mathcal{C}(t_i), i=1,2,...,f-1} \sum_{i=0}^{f-1} f(x_i, x_{i+1})$$

where

$$f(x_i, x_{i+1}) = (Dx_i - Ex_{i+1})^TW^{-1}(Dx_i - Ex_{i+1})$$

$$+ (x_i - x_{i+1})^TP(x_i - x_{i+1})$$

$$+ 2(x_i - x_{i+1})^T \bar{A}^TPA x_{i+1} - hr_i$$

and

$$r_i = -x_i^{T}A^ TP^{-1} QP^{-1} \bar{A}^TPA x_{i+1}.$$
4. Restatement of the problem

Ax_{i+1} is obtained and 
\[ J_i(x_i, u_i, x_{i+1}) = \int_{t_i}^{t_{i+1}} \tilde{x}_i^T(t)Q\tilde{x}_i(t) + u^T(t)R u(t) dt. \]

Now, letting
\[
\begin{align*}
\dot{A}_i &= \begin{bmatrix} A & A x_{i+1} \\ 0 & 0 \end{bmatrix}, \\
\dot{B}_i &= \begin{bmatrix} B \\ 0 \end{bmatrix}, \\
\dot{Q}_i &= \begin{bmatrix} Q & 0 \\ 0 & r_i \end{bmatrix}, \\
\dot{P}_i &= \begin{bmatrix} P & q_i \\ q_i^T & 0 \end{bmatrix}, \\
q_i &= -\tilde{A}^T P A x_{i+1}, \\
\tilde{x}_i &= \begin{bmatrix} x_i \\ 1 \end{bmatrix},
\end{align*}
\]

\[ \dot{P}_i \dot{A}_i + \dot{A}_i^T \dot{P}_i - \dot{P}_i \dot{B} R^{-1} \dot{B}^T \dot{P}_i + \dot{Q}_i = 0 \]
can be obtained. This leads to
\[
\begin{align*}
J_i(x_i, u_i, x_{i+1}) &= \int_{t_i}^{t_{i+1}} \left[ \tilde{x}_i^T(t) \dot{Q}_i \tilde{x}_i(t) + u^T(t) R u(t) - r_i \right] dt \\
&= \tilde{x}_i^T(t_i) \dot{P}_i \tilde{x}_i(t_i) - \tilde{x}_i^T(t_{i+1}) \dot{P}_i \tilde{x}_i(t_{i+1}) \\
&\quad + \int_{t_i}^{t_{i+1}} u_i^T(t) R u_i(t) dt - h_i
\end{align*}
\]

where
\[
\begin{align*}
u_i(t) &= u(t) + R^{-1} \dot{B}^T \dot{P}_i \tilde{x}_i \\
&= u(t) + R^{-1} B^T P (x(t) - x_{i+1}) - R^{-1} B^T \tilde{A} \tilde{x}_{i+1}.
\end{align*}
\]

Since \( x(t_i) = x_i, \tilde{x}_i(t_i) = x(t_i) - x(t_{i+1}) = x_i - x_{i+1} \) and \( \tilde{x}_i(t_{i+1}) = x(t_{i+1}) - x(t_{i+1}) = 0 \),
\[
J_i(x_i, x_{i+1}, u_i) = (x_i - x_{i+1})^T P (x_i - x_{i+1}) \\
-2(x_i - x_{i+1})^T \tilde{A} \tilde{x}_{i+1} \\
+ \int_{t_i}^{t_{i+1}} u_i^T(t) R u_i(t) dt - h_i
\]

Thus the problem is reduced to the minimum energy control problem with respect to \( u_i \); for given \( x_i \) and \( x_{i+1} \), i.e. find a control \( v_i^* \) satisfying \( (x(t_i), x(t_{i+1})) = (x_i, x_{i+1}) \) and minimizing the cost \( \dot{J} = \int_{t_i}^{t_{i+1}} u_i^T(t) R u_i(t) dt \), in which the optimal controller and the minimum cost are given by
\[
\begin{align*}
v_i^*(t) &= -R^{-1} B^T e^{\tilde{A}^T(t_{i+1} - t)} W^{-1} (D x_i - E x_{i+1}) \\
\min \dot{J} &= (D x_i - E x_{i+1})^T W^{-1} (D x_i - E x_{i+1}),
\end{align*}
\]

respectively [2, 6]. It is remarked here that \( W \) is a positive-definite matrix given by \( W = \int_0^h e^{\tilde{A}^T(t_{i+1} - \tau)} B R^{-1} B^T e^{\tilde{A}^T \tau} d\tau \), which is a solution to (1.5) (see Appendix A). \( \Box \)
4.3 Calculation of the optimal intermediate target states

Theorem 4.1 gives the optimal input $u^*(t)$ in an explicit form including the intermediate target states as parameters, which are the solution of the problem given in (4.5). It also turns out from the proof that the value of (4.5) gives the minimum value of the cost function (4.1), because

$$J(x_0, u^*, \{x_i\}_{i=1,2,...,f-1}) = \sum_{i=0}^{f-1} f(x_i, x_{i+1}). \quad (4.6)$$

and the solution to (4.5) leads to the optimal intermediate target states $x_i^*$. To solve this problem, again the BFS algorithm, presented in chapter 2 can be used. Note however, that because of the discretization of state space and time-axis, the found solution is still a sub-optimal solution to the original problem, Problem 1.1.
Chapter 5

Numerical simulations of the final result

In this chapter the effectiveness of the control strategy derived in the previous chapter will be tested using numerical simulations. At the end of this chapter, the computation time required to solve the problem will be discussed.

5.1 A simple example

Consider again the simple vehicle of Fig.3.1 where, if \( m = 1 \), the dynamics are given by:

\[
\dot{x} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u, \quad \text{where} \quad x = \begin{bmatrix} x_p \\ x_v \end{bmatrix}
\]

where \( x_p \) is the position of the vehicle motion restricted to one dimensional direction and \( x_v \) is its velocity. Suppose that the system is moving through the same field with randomly distributed obstacles with constant velocity in the direction perpendicular to \( x_p \). The discretized region of \( x_p \) consists of a set of 10 discrete points at each time \( t_i \), and the distribution of the obstacles in the grid is random. The discrete time interval \( h = 1 \) s. Now, the discretization of the velocity \( x_v \) consists of a set of 11 velocities varying from \(-1.0\) to \(1.0\) with an interval of \(0.2\). Again, for simplicity, the maximum motion of the system is restricted to one grid point at each time-step, which means that the number of edges from a node to the nodes at the next time-step is less than or equal to 3. At each time-step \( t_i \) the optimal input \( u^*(t) \) is computed for 20 prediction steps, i.e, \([t_i, t_{i+20}]\), and \( u^*(t) \) is applied only for \( t \in [t_i, t_{i+1}] \). Furthermore, the final state \( x_f \) is not fixed in each optimal control problem here, \( x_f \) is also optimized in the same way as \( x_i \),
5. Numerical simulations of the final result

Figure 5.1: Numerical simulation results for the undamped system

\[ i = 1, 2, \ldots, f - 1, \text{ in } (4.5). \] The weighting matrices of the cost function (4.1) are given by

\[ Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad R = 1 \]

Fig.5.1 shows the results of the first 40 steps.

What can be seen in the figure is that the system moves one way until it encounters an obstacle. This seems not the optimal path through the field because it is not the shortest path possible. However, there is no friction in the system. It can move frictionless, so a control input is only needed to accelerate or decelerate the system. The path found is the path that requires the least control energy and therefore it is the optimal path. In fact, if the optimal path will be computed for 40 steps, the same result will be found.

Now let us add some viscous damping to the system. In that case, the dynamics of the system are given by:

\[ \dot{x} = \begin{bmatrix} 0 & 1 \\ 0 & -10 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u, \quad \text{where } \quad x = \begin{bmatrix} x \\ \dot{x} \end{bmatrix} \]

All the other conditions are the same as in the previous case. The results are presented in Fig.5.2. The resulting path is the path that requires least control energy and, as one would expect in the case of the damped system, also the shortest possible path through the field with respect to distance.
5. Numerical simulations of the final result

Figure 5.2: Numerical simulation results for the damped system

5.2 Performance of the used algorithm

Fig.5.3 shows the computational costs of the used BFS algorithm, which is coded in C and has been running on a single PC with a Pentium IV 3.06 GHz CPU and 2.0 GB RAM. The used BFS implementation can be found in Appendix B. The figure shows the computational costs for a worst case scenario. The discrete set that has been searched does not contain any obstacles, so each discrete point in the set has to be visited. In this case, again the displacement of the system is restricted to one grid point each time-step. Furthermore the costs have been calculated for up to 50 prediction steps and several discrete velocity sets.

The low computation time of the algorithm makes it possible to use this control strategy in a real time application. Further enlargement of the number of discrete velocities of \( x_n \), the number of prediction steps or the number of nodes that can be addressed from each single node may contribute to a better performance of the controller. The computation time however, will increase rapidly as can be depicted from the relation derived in chapter 2. There exists a trade off between the computation time and the other parameters such as the number of prediction steps. Since the BFS algorithm used in this approach is a forward search algorithm, this approach can be used even if the maximum allowable computation time is fixed, although the number of prediction steps may vary. This is one advantage of this approach.
5. Numerical simulations of the final result

Figure 5.3: Worst case computational costs of the BFS Algorithm for several discrete velocity sets
Chapter 6

Conclusions and recommendations

6.1 Conclusions

In this report, the optimal control problem subject to non-convex state constraints has been addressed and is applied to the obstacle avoidance problem. To relax the strong state constraints the method of discretizing time-axis and state space has been used. The problem has been stated using a usual continuous-time cost function. A solution to this problem has been obtained. The optimal input for this problem has been derived in an explicit form, including the optimal next discrete state as a parameter. The optimal next discrete state can be obtained by solving a discrete minimization problem. To solve this discrete minimization problem, an efficient numerical algorithm, the Breadth First Search (BFS) algorithm has been presented. After pointing out, using numerical simulations, that the use of the usual continuous-time cost function does not give satisfactory results if non-convex state constraints are set, a new cost function has been proposed that includes intermediate desired target points. This cost function makes it possible to optimize the intermediate target states and the continuous-time behavior simultaneously. The optimal control for this new problem has been derived in an explicit form, including the intermediate target states as parameters. The intermediate target states have been obtained by solving a discrete optimization problem, using the BFS algorithm. Furthermore, numerical simulations have been performed to proof the effectiveness of the approach.
6.2 Recommendations

Future research topics could include:

- The optimal way to discretize the state space, to maximize the efficiency of the derived control strategy. Especially in the higher dimensional case, the computation time will increase rapidly, so the need to use more sophisticated sampling methods and state space discretizations becomes evident.

- Although the research presented in this report mainly focused on the obstacle avoidance problem, the proposed approach can be extended to the more general class of the optimal control problem with non-convex state constraints, such as the control problem of hybrid systems. Note however, that the proposed approach can only be applied to (piecewise) linear systems.
Bibliography


Appendix A

Calculation of $W$

Lemma A.1

\[ W = \int_{t_i}^{t_{i+1}} e^{A(t_i - \tau)} B R^{-1} B^T e^{A^T(t_i - \tau)} d\tau \]  

(A.1)

is a solution of:

\[ W A^T + A W - e^{A(t_i - t_{i+1})} B R^{-1} B^T e^{A^T(t_i - t_{i+1})} + B R^{-1} B^T = 0 \]  

(A.2)

Proof:

\[
\begin{align*}
W A^T + A W &= e^{A(t_i - t_{i+1})} B R^{-1} B^T e^{A^T(t_i - t_{i+1})} - B R^{-1} B^T \\
&= \int_{t_i}^{t_{i+1}} e^{A(t_i - \tau)} B R^{-1} B^T e^{A^T(t_i - \tau)} A^T + A e^{A(t_i - \tau)} B R^{-1} B^T e^{A^T(t_i - \tau)} d\tau \\
&= \int_{t_i}^{t_{i+1}} \frac{d}{d\tau} \left[ e^{A(t_i - \tau)} B R^{-1} B^T e^{A^T(t_i - \tau)} \right] d\tau \\
&= \left[ e^{A(t_i - \tau)} B R^{-1} B^T e^{A^T(t_i - \tau)} \right]_{t_i}^{t_{i+1}} \\
&= e^{A(t_i - t_{i+1})} B R^{-1} B^T e^{A^T(t_i - t_{i+1})} - B R^{-1} B^T 
\end{align*}
\]
A. Calculation of $W$
Appendix B

Breadth First Search Algorithm

This appendix gives a code listing of the BFS algorithm used for the numerical simulations in this report. The algorithm is programmed in C for use on a windows platform.

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <float.h>

#define steps 20
#define b (2*steps+1)
#define l (steps+1)
#define speedlength 5
#define startnode 3

bool FALSE = 0;
bool TRUE = 1;
int *data;
long btot;
long ltot;
long newnode = startnode;
long linecount = 0;
long stackpointer = 0;
long graph[b*l];
long visit[b*l*speedlength];
long route[b*l];
long stack[2*b*speedlength];
float newspeed=0.0;
float speed[b*l];
float speedstack[2*b*speedlength];
```
float cost[11];
float speedset[speedlength] = {-1.0, -0.5, 0.0, 0.5, 1.0};
double comptime = 0;
double totaltime = 0;
double avgtime = 0;

void readstr(FILE *f, char *string)
{
    do
    {
        fgets(string, 255, f);
    } while ((string[0] == '/') || (string[0] == ' '));
    return;
}

void SetupGraph(void)
{
    int datasample;
    FILE *filein;
    char oneline[255];

    filein = fopen("obstacles.txt", "rt");
    readstr(filein, oneline);
    sscanf(oneline, "length %d width %d", &l, &b);
    data = (int*)malloc(l*b*sizeof(int));
    for (int loop = 0; loop < l*b; loop++)
    {
        readstr(filein, oneline);
        sscanf(oneline, "%d", &datasample);
        data[loop] = datasample;
    }
    fclose(filein);
}

void makeSubGraph(long point)
{
    long counter = 0;
    for(long r=1;linecount;r<liencount+1;r++)
    {
        for(long c = point-steps;c <= point+steps;c++)
        {
            int sample = 0;
            if(c < 0 || c >= b || r >= l)
            {
                sample = 1;
            }
            else {sample = data[r*b+c];}
            graph[counter] = sample;
        }
    }
}
B. Breadth First Search Algorithm

counter++;  
}
}

void addToStack(long node, float speed)
{
    stack[stackpointer] = node;
    speedstack[stackpointer] = speed;
    stackpointer++;
}

void removeFromStack(void)
{
    int i;
    for(i=0; i<2*b*speedlength; i++)
    {
        stack[i] = stack[i+1];
        speedstack[i] = speedstack[i+1];
    }
    stackpointer--;  
}

void initSearch(void)
{
    addToStack(steps, newspd);
    long i;
    for(i=0; i<b; i++)
    {
        cost[i] = 1000000000.0;
        graph[i] = 0;
    }
    for(i=0; i<b*length; i++)
    {
        visit[i] = 0;
    }
    cost[steps] = 0.0;
    speed[steps] = 0.0;
}

void freeMemory(void)
{
    free(graph);
    free(visit);
    free(stack);
    free(speedstack);
    free(route);
    free(cost);
    free(speedset);
B. Breadth First Search Algorithm

```c
free(speed);
free(data);
}

float calculateCosts(unsigned int xx1, float y1, unsigned int xx2, float y2)
{
    int x1;
    int x2;
    x1 = xx1%b;
    x2 = xx2%b;
    float cost;
    cost = 10.1489*(x1-x2)*(x1-x2)+2.0*(x1-x2)*(y1-y2)+0.1489*(y1-y2)*(y1-y2) -
        2.0*(-2.0*y2*(x1-x2))+2.0*0.1489*y2+ (y1-y2) +
        (125.9467*x1-125.9467*x2+12.5318*y1+11.8761*y2) *
        (0.9143*x1-0.9143*x2+0.091*y1)+
        (-12.5318*x1+12.5318*x2-1.246*y1-21.4492*y2)+
        (-0.091*x1+0.091*x2-0.009*y1-y2) +y2*y2;
    return cost;
}

long minIndex(float *arr)
{
    long indx = 0;
    float high = 1000000000.0;
    for(long i=0;i<b;i++)
    {
        if(arr[i] < high)
        {
            high = arr[i];
            indx = i;
        }
    }
    if(high == 1000000000.0)
    {
        return -1;
    }
    return indx;
}

int checkNode(long node, int type)
{
    if((graph[node+b+type] == 1 || node+b+type >= (b)*(1))
    {
        return FALSE;
    }
    return TRUE;
}

bool Search(void)
```

34
B. Breadth First Search Algorithm

```c
double start = (clock() * 1000)/CLOCKS_PER_SEC;
while(stackpointer > 0)
{
    int speedcnt = speedlength;
    long node;
    float spd;
    node = stack[0];
    spd = speedstack[0];
    if (checkNode(stack[0], -1))
    {
        int i;
        for(i=0; i<speedcnt; i++)
        {
            float costs, totalcosts;
            costs = calculateCosts(node, spd, node+b-1, speedset[i]);
            totalcosts = cost[node] + costs;
            if (totalcosts < cost[node+b-1])
            {
                cost[node+b-1] = totalcosts;
                route[node+b-1] = node;
                speed[node+b-1] = speedset[i];
            }
            if (visit[(b+1*i+node+b-1)] == 0)
            {
                addToStack(node+b-1, speedset[i]);
                visit[(b+1*i+node+b-1)] = 1;
            }
        }
    }
    if (checkNode(stack[0], 0))
    {
        int i;
        for(i=0; i<speedcnt; i++)
        {
            float costs, totalcosts;
            costs = calculateCosts(node, spd, node+b, speedset[i]);
            totalcosts = cost[node] + costs;
            if (totalcosts < cost[node+b])
            {
                cost[node+b] = totalcosts;
                route[node+b] = node;
                speed[node+b] = speedset[i];
            }
            if (visit[(b+1*i+node+b)] == 0)
            {
                addToStack(node+b, speedset[i]);
                visit[(b+1*i+node+b)] = 1;
            }
        }
    }
}
```
B. Breadth First Search Algorithm

```c
if (checkNode(stack[0]+1))
{
    int i;
    for(i=0; i<speedcnt; i++)
    {
        float costs, totalcosts;
        costs = calculateCost(node, spd, node+b+1, speedset[i]);
        totalcosts = cost[node]+costs;
        if(totalcosts < cost[node+b+1])
        {
            cost[node+b+1] = totalcosts;
            route[node+b+1] = node;
            speed[node+b+1] = speedset[i];
        }
        if(visit[(b*l*i+node+b+1)] == 0)
        {
            addToStack(node+b+1, speedset[i]);
            visit[(b*l*i+node+b+1)] = 1;
        }
    }
}
removeFromStack();
}
long until;
if(linecount+steps >= tot)
{
    until = tot-linecount;
}
else until = steps;
float finalrow[b];
for(int k=0; k<b; k++)
{
    finalrow[k] = cost[b*until+k];
}
long index;
index = minIndex(finalrow);
if(index == -1)
{
    return FALSE;
}
index = b*until+index;
long routerow[steps];
routerow[until] = index;
for(int s=(until-1); s>=0; s--)
```
B. Breadth First Search Algorithm

```c
{    
    index = route[index];
    routerow[s] = index;
}
long tempnode;
    tempnode = routerow[1];
newnode = newnode+tempnode-b-steps;
newspeed = speed[tempnode];
double end = (clock()*1000)/CLOCKS_PER_SEC;
comptime = end-start;
return TRUE;
}

void main(void)
{
    SetupGraph();
    FILE *fidtime;
    FILE *fiddata;
    fidtime = fopen("time.txt","w");
    fiddata = fopen("data.txt","w");
    SetupGraph();
    while(linecount < ltot-1)
    {
        initSearch();
        makeSubGraph(newnode);
        bool test = TRUE;
        test = Search();
        if(test == FALSE)
        {
            printf("ERROR: No feasible solution found in step \%d \\
                        \n",linecount);
            break;
        }
        totaltime = totaltime + comptime;
        avgtime = totaltime/(linecount);
        linecount++;
        fprintf(fidtime,"\%d \t \%f \n",linecount,comptime);
        fprintf(fiddata,"\%d \t \%f \n",newnode,newspeed);
    }
    fprintf(fidtime,"\nDatapoints: \%d \t Total speed: \%f msecs \t \
                                Average speed: \%f msecs \n\nEnd of File",linecount,totaltime,avgtime);
    fclose(fidtime);
    fclose(fiddata);
    freeMemory();
    printf("Program terminated \nPress enter to quit");
    getchar();
}
```
B. Breadth First Search Algorithm
Appendix C

Alternative numerical search algorithms

C.1 Dijkstra’s algorithm

Dijkstra’s algorithm is a shortest path algorithm that finds the shortest path to every vertex in a graph. The algorithm is based on building a tree that gives the shortest path to all vertices in the searched graph, the Dijkstra tree. To find the shortest path to any of the vertices starting from the initial vertex, one can simply follow the tree to the desired vertex.

Because of these properties, Dijkstra’s algorithm runs in logarithmic time $O(m \log n)$, where $m$ is the number of edges and $n$ the number of vertices in the graph.

Dijkstra’s algorithm is typically implemented using a heap (as opposed to the queue used in the breadth first search method). To build the Dijkstra tree, one starts from a starting vertex adding edges to a tree, each time step, choosing the shortest possible edge to add.

One main drawback of Dijkstra’s algorithm is that it cannot deal with negative weights in the graph.

C.2 A* algorithm

The A* algorithm is an extension of the Breadth First Search method. It is a graph search algorithm that finds the shortest path from an initial vertex to a given goal vertex. The main difference between the A* algorithm and breadth first search is that the A* algorithm uses a priority queue. It employs a heuristic test which ranks each vertex by an estimate of the shortest edge to the next vertex. It visits the nodes in order of this heuristic...
C. Alternative numerical search algorithms

Because of the heuristic test, the computational cost of the A* algorithm are higher than the computational cost of the breadth first search method. The advantage of the A* algorithm however is, that not all the vertices have to be visited to find the shortest path to a desired vertex, which can decrease the computational amount. For large paths however, this advantage will be lost because the construction of the priority queue will add up to the computational costs considerably.
Appendix D

Article submitted to the IEEE CCA '04
Approximate Continuous-time Optimal Control in Obstacle Avoidance by Time/Space Discretization of Non-convex State Constraints

H.L. Hagenaars  
Department of Mechanical Engineering  
Eindhoven University of Technology  
The Netherlands  
h.l.hagenaars@student.tue.nl

J. Imura  
Graduate School of Information Science and Engineering  
Tokyo Institute of Technology  
Japan  
imura@cyb.mei.titech.ac.jp

H. Nijmeijer  
Department of Mechanical Engineering  
Eindhoven University of Technology  
The Netherlands  
h.nijmeijer@tue.nl

Abstract— This paper addresses an approximate version of the optimal control problem with non-convex state constraints via discretization of time and space, where the specific application pursued is the obstacle avoidance problem. First, it is pointed out that the standard continuous-time cost function with the final state fixed is not suitable to the optimal control problem under the non-convex state constraints, and then a new cost function including intermediate target states is proposed. Next, for the optimal control problem with this cost function, where the non-convex state constraints are discretized with respect to time axis and state space, an optimal continuous-time control is given in an explicit form, including the intermediate target states obtained by solving a discrete optimization problem. Efficient algorithms such as the Breadth First Search algorithm can be applied to this discrete problem. Thus the continuous-time trajectory as well as the discretized state at each discrete time are simultaneously optimized. Finally, we illustrate the effectiveness of the proposed approach with numerical simulations.

I. INTRODUCTION

The problem of planning an optimal trajectory of a mobile robot avoiding (possibly moving) obstacles is one of the challenging topics in the field of mobile robotics, since it belongs to the class of optimal control problems of continuous-time dynamical systems under a non-convex constraint set and hence it is very difficult to rigorously solve it.

One realistic approach is to utilize the discretization technique of time axis and state space to formalize the collision avoidance problem in terms of the discrete optimization problem, which allows to apply efficient algorithms such as the Dijkstra method and the A* method (e.g., [1], [2], [3]). However, since the minimization of the discrete-time cost function on the discrete set of states is considered in this approach, a continuous-time trajectory that interpolates between the obtained optimal discretized states will have to be calculated for the implementation. Hence, we cannot simultaneously optimize the continuous-time behaviors as well as the discretized states.

Another well-known approach is to add some potential functions for avoiding the obstacles to the original cost function; thus the collision avoidance problem is described as an optimal problem without constraints (e.g., [4], [5]). Since the optimal problem in this case includes no constraints, the gradient method is mainly used, which results in the real-time optimization. However, the obtained solution is, in general, a local minimum, and the algorithm may fall into some deadlock even if an optimal solution exists. So how to properly construct the potential functions is one of the significant issues in this approach; however, it will not be so easy to solve this in a general framework.

This paper follows the same basis as the former approach, i.e., the discretization of time axis and state space, and proposes an approach that the discretized state at each discrete time and the continuous-time behavior between these discretized states are simultaneously optimized in the collision avoidance problem. First, it is pointed out that the standard continuous-time cost function with the final state fixed is not suitable to the non-convex optimal control problem such as the collision avoidance problem, and then a new type of cost function including intermediate target states is introduced. Next, to solve this problem in an approximate but efficient way, the non-convex constraints are relaxed via discretization of time and space, while the continuous-time system and the cost function are not discretized. For the problem discretized in this way, a continuous-time control and intermediate target states minimizing this new cost function are derived; an optimal continuous-time control is given in an explicit form including the intermediate target states as parameters, and optimal intermediate target states are obtained by solving some discrete optimization problem, for which efficient algorithms such as the Breadth First Search (BFS) algorithm or equivalently the Forward Dynamic Programming (FDP) can be applied. Thus the continuous-time behavior and the discretized state at each discrete time of the continuous-time system can be simultaneously optimized for the discretized problem. Finally, numerical simulations show the effectiveness of the proposed approach.

II. PROBLEM STATEMENT

Fig.1 gives a simple two dimensional example of an obstacle avoidance problem. Our goal is to obtain a piecewise continuous input \( u(t) \) that drives a mobile robot, starting in point \( \mathbf{x}_0 \), to the final state \( \mathbf{x}_f \) using optimal control theory. However, due to the obstacles present in the field, the
Fig. 1. A simple 2 dimensional obstacle avoidance problem

state \( x(t) \) of the dynamical system considered cannot take arbitrary values, that is, the state \( x(t) \) must be included in a constraint set \( C(t) \), where \( C(t) \) is some given time-invariant or time-varying, non-convex subset of \( \mathbb{R}^n \).

Such a problem may be formulated as follows.

**Problem 2.1:** Given the initial and final states \( (x_0, x_f) \), find \( u(t) \in \mathcal{P}C^m \), where \( \mathcal{P}C \) is the set of all piecewise continuous functions, minimizing

\[
J(x_0, u) = \int_{t_0}^{t_f} \left[ (x(t) - x_f)^T Q (x(t) - x_f) + u^T(t) R u(t) \right] dt
\]

subject to:

\[
\dot{x} = Ax + Bu
\]

\[
x \in \mathbb{R}^n, u \in \mathbb{R}^m
\]

\[
x(t) \in C(t), \forall t \in [t_0, t_f], \quad x(t_f) = x_f
\]

where \( C(t) \) is a given time-invariant or time-varying, non-convex subset of \( \mathbb{R}^n \), and \( Q \geq 0, R > 0 \). Furthermore we assume the pair \( (A, B) \) is controllable.

The cost function stated in Problem 2.1 is known from the standard finite-time optimal control problem with both initial and final states fixed, where the constraint set \( C(t) \) is convex \([6]\). However, in the optimal control problem with a non-convex constraint set, we can easily point out that if the above cost function is used, it does not lead to the desired result. Fig.2 illustrates the encountered problem. The dashed line indicates the optimal path from \( x_0 \) to \( x_f \) that will be obtained using the continuous-time cost function stated in Problem 2.1. While driving the system from \( x_0 \) to \( x_f \) the controller will minimize the distance to the final target \( x_f \) at each time, which will result in the illustrated phenomenon. On the other hand, the solid line in Fig.3 indicates the optimal path we would like to achieve. Thus it will be required to take an other more suited cost function in order to produce an outcome closer to what is 'expected' or desired. One way to do so is to introduce, as shown in Fig.3, the intermediate target states \( x'_f, x^*_f \) and \( x_f \) to be addressed into the cost function. Such a change in the cost function gives us the ability to optimize the intermediate target states and the trajectory in the time interval simultaneously. Although another way is to add a proper potential function to the cost function, it will be difficult to find it in a systematic way because we have to specify e.g., the desired trajectory between \( x_0 \) and \( x'_f \) in Fig.3 in an indirect way by using some potential function, instead of using the intermediate target states.

The above point leads to the following new problem formulation that includes the proposed cost function.

**Problem 2.2:** Given the initial and final states \( (x_0, x_f) \). Find \( u(t) \in \mathcal{P}C^m \) and intermediate target states \( x_i \in \mathbb{R}^n, i = 1,2,\ldots,f-1 \), minimizing

\[
J(x_0, u, \{x_i\}_{i=1,2,\ldots,f}) = \sum_{i=0}^{f-1} \int_{t_i}^{t_{i+1}} \left\{ (x(t) - x_{i+1})^T Q (x(t) - x_{i+1}) + u^T(t) R u(t) \right\} dt
\]

subject to:

\[
\dot{x} = Ax + Bu
\]

\[
x(t_i) = x_i \in C_d(t_i), \quad i = 0,1,\ldots,f
\]

where \( t_i = ih, h \) denotes the discrete time interval, \( C_d(t_i) \) is a time-invariant or time-varying set of discrete points in \( \mathbb{R}^n \) given by discretizing the non-convex constraint set \( C(t_i) \) in Problem 2.1 in some appropriate way such as the gridding method, \( Q \geq 0, R > 0 \), and the pair \( (A, B) \) is controllable.
In this problem, the intermediate target states $x_i, i = 1, 2, \ldots, f - 1$, are introduced in the cost function of (2), and further are optimized together with the input so as to minimize the cost function, which will enable us to obtain the desired trajectory as in Fig.3. Furthermore, it is difficult to guarantee that the strong constraint $x(t) \in C(t)$ will hold at any time as in Problem 2.1. So in Problem 2.2, the non-convex constraints are relaxed by discretization of time and space, that is, we suppose that the discrete set $C_d(t_i)$ is given by discretizing the continuous set $C(t)$ in some way, and that the constraint condition is satisfied only at every discrete time $t_i$, i.e. $x(t_i) \in C_d(t_i)$. In this way, the discretization of time and space is used for relaxing the non-convex constraints, although a solution to Problem 2.2 is sub-optimal for the original problem. Note also that, although this problem is an optimal control problem with the terminal state $x_f$ fixed, $x_f$ can be also optimized in the same way as $x_i, i = 1, 2, \ldots, f - 1$, if the problem with $x_f$ free is considered.

III. PROBLEM SOLUTION

In this section we will provide a solution to the problem stated in the previous section. We will derive an optimal control in an explicit form including the intermediate target state as a parameter. Furthermore we will show that an efficient algorithm such as Forward Dynamic Programming (FDP) or equivalently Breadth First Search (BFS) algorithm can be applied to obtain the optimal intermediate target states.

A. Derivation of optimal controller

A solution to Problem 2.2 is given as follows.

**Theorem 3.1:** For Problem 2.2, an optimal control $u^* \in \mathcal{U}$ is given by

$$u^*(t) = -R^{-1}BP(x(t) - x_{i+1})^T + R^{-1}B^TA^TPAz_{i+1}$$

$$-R^{-1}B^Te^{At}W^{-1}(Dx_i - Ex_{i+1})$$

where $P$ and $W$ are the positive-definite solutions of

$$PA + A^TP - PBRA^{-T}B^TP + Q = 0$$

$$WA^T + A^W + BR^{-1}B^T - e^{Ah}BR^{-1}B^Te^{AT} = 0$$

(respectively, and

$$\dot{A} = A - BR^{-1}B^TP, \ D = e^{Ah},$$

$$E = \int_0^h e^{Ah} d\tau (I + BR^{-1}B^TA^{-T}P)A - e^{Ah},$$

and $x_i, i = 1, 2, \ldots, f - 1$, are the solutions to the following discrete optimization problem.

$$\min_{x_i \in C_d(t_i), i = 1, 2, \ldots, f - 1} \sum_{i=0}^{f-1} f(x_i, x_{i+1})$$

where

$$f(x_i, x_{i+1}) = (Dx_i - Ex_{i+1})^TW^{-1}(Dx_i - Ex_{i+1})$$

$$+(x_i - x_{i+1})^TP(x_i - x_{i+1})$$

$$+2(x_i - x_{i+1})^T \tilde{A}^TPAx_{i+1} - hr_i$$

and

$$r_i = -x_{i+1}^T \tilde{A}^TPAx_{i+1} - hr_i.$$
Theorem 3.1 gives the optimal input $u^*(t)$ in an explicit form including the intermediate target states as parameters, which are the solution of the problem (6). It also follows from the proof that the value of (6) gives the minimum value of the cost function (2), because

$$J(x_0, u^*, \{x_t\}_{t=1,\ldots,f}) = \sum_{i=0}^{f-1} f(x_t, x_{t+1}).$$

and the solution to (6) leads to the optimal intermediate target states $x^*_t$.

### B. Calculation of optimal intermediate target states

Let us next discuss how to solve the problem (6). This problem is equivalent to finding the shortest path in a graph shown in Fig.4 (suppose $x_t \in \mathbb{R}$), where the discretized state space can be represented as a graph whose nodes are the points in $C_d(t)$. So we can apply the Dijkstra algorithm or A* algorithm to this problem. However, since the graph in Fig.4 has a hierarchical structure caused by the time series, we will exploit it to solve the problem (6) in a more efficient way.

Define

$$V_k(x_k) := \min_{x_{i+1} \in \mathbb{R}} \sum_{i=0}^{k-1} f(x_i, x_{i+1}).$$

So we have the relation

$$V_{k+1}(x_{k+1}) = \min_{x_k} \{f(x_k, x_{k+1}) + V_k(x_k)\}.$$  

From this relation it turns out that the Forward Dynamic Programming (FDP) approach can be applied to solve the problem (6). Furthermore, we can interpret this approach as the Breadth First Search (BFS) algorithm, cf. [8]. So the computational effort needed for solving the problem is given by $O(n_e)$, where the total number of the edges in the graph in question is $n_e$.

Consider the case of the mobile robot, which can move only in one direction. So the state of the system is given by the position $x_p \in \mathbb{R}$ and the velocity $x_u \in \mathbb{R}$. Suppose that the position region is restricted by several obstacles and the velocity is not restricted. Fig.5 shows a part of the graph that represents the discretized space in this case. Each discrete position $x_p$ contains two discrete velocities $x_u$. The gray discrete position cannot be addressed because of the presence of an obstacle at that node, which implies that the corresponding edges (dashed lines) are taken off. Let $f$ denote the number of prediction steps, $n_u$ denote the number of discrete velocities at each $x_p$, $n_v$ denote the total number of the nodes including the nodes in which the obstacles exist in the graph for the whole prediction steps, $n_{out}$ denote the maximum number of edges outgoing from each node of discrete positions, and $n_e$ denote the number of edges in the graph (not including the nodes in which the obstacles exist). In the case of Fig.5, for each position $x_p$, 2 discrete velocities $x_u$ can be addressed, so in this case $n_u = 2$. Each position $x_p$ is connected forward to three other nodes of discrete positions and therefore $n_{out} = 3$. In
the general case, the total number of nodes is given by

\[ n_n = n_n 0 \sum_{k=0}^{f-1} (k(n_{out} - 1) + 1) \]

\[ = n_n 0 \left( \frac{1}{2}n_{out}^2 - 1 + \frac{1}{2}(n_{out} + 1)f + 1 \right) \]

So the total number \( n'_e \) of edges in the graph with \( n_n \) nodes in the absence of obstacles is

\[ n'_e = n_n 0 \sum_{k=0}^{f-1} (k(n_{out} - 1) + 1). \]

On the other hand, we do not need to in general consider 2\( n_n 0 n_{out} \) edges for each obstacle; for example, the edges lost in Fig.5 due to the presence of the obstacle are represented by the dashed lines going towards and leaving the obstacle. Let \( P \in [0, 1] \) denote the uniform probability density for emergence of obstacles in the prediction area. Then if \( P \) is not so large \( (P < 0.5 - \varepsilon \) for some positive number \( \varepsilon \)), the number of edges is approximately given by

\[ n_e = n'_e - 2Pn'_e 0 n_{out} \sum_{k=0}^{f-1} (k(n_{out} - 1) + 1) \]

\[ - Pn'_e 0 n_{out} \{f(n_{out} - 1) + 1\} \]

\[ = (1 - 2P)n'_e - Pn'_e 0 n_{out} \{f(n_{out} - 1) + 1\}. \tag{9} \]

The last term in this equation represents the number of edges lost due to the obstacles present in the last prediction step. As we can see in Eq.9, the number of edges decreases proportionally to increasing probability density \( P \) of the obstacles. Consider the case \( f = 10, n_e = 1 \) and \( n_{out} = 3 \). If there are no obstacles present, i.e., \( P = 0 \), the number of edges \( n_e \) will be 300. However, if \( P = 0.2 \) and \( P = 0.3 \), the number of edges decreases to \( n_e = 168 \) and \( n_e = 101 \), respectively.

Note also that in the above approach the discretization applies to the whole state space \( C(t) \), not only to a part of the state space such as the position space. Indeed, it will be useful that the discretization for only a part of the state space specified by non-convex constraints (e.g., the position space including obstacles) can be applied and no discretization for the other free space (e.g., the velocity space without constraints) is applied. However, in this case, the above FDP (or BFS) algorithm cannot be directly applied. Such an extension is left for future research.

IV. NUMERICAL SIMULATION

To test the effectiveness of the derived control strategy, we will show some simulation results. At the end of this section we will discuss the computation time required to solve the problem.

Consider a simple vehicle given by

\[ \dot{x} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u, \text{ where } x = \begin{bmatrix} x_p \\ x_v \end{bmatrix} \]

where \( x_p \) is the position of the vehicle motion restricted to one dimensional direction and \( x_v \) is its velocity. On the other hand, we suppose that the vehicle is moving through a field with obstacles, at constant velocity in the direction perpendicular to the \( x_p \) axis. The discretization of the position \( x_p \) consists of a set of 10 discrete points at each time \( t_i \), and the distribution of the obstacles in the grid is random. The discretization of the velocity \( x_v \) consists of a set of 11 velocities varying from -1.0 to 1.0(m/s) with an interval of 0.2. For simplicity the maximum movement of the system is restricted to one grid point at each time-step, that is, the number of edges from a node to the nodes at the next time-step is less than or equal to 3. A receding horizon policy is used for control of the vehicle, that is, at time-step \( t_i \) the optimal control \( u^*(t) \) is computed for 20 prediction steps, i.e., \( [t_i, t_{i+20}] \), and \( u^*(t) \) is applied only for \( t \in [t_i, t_{i+1}] \). Furthermore, we do not fix the final state \( x_f \) in each optimal control problem here, \( x_f \) is also optimized in the same way as \( x_v \), i.e., \( i = 1, 2, \ldots, f - 1 \), in (6). The weighting matrices \( Q \) and \( R \) of the cost function (2) is given by

\[ Q = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad R = 1. \]

Then Fig.6 shows the results of the first 40 steps.

What we see in the figure is that the system moves one way until it encounters an obstacle. This seems not the optimal path through the field because it is not the shortest path possible. However, the system is not damped. It can move frictionless, so an input is only needed to accelerate or decelerate the system. The path found is the path that requires the least control energy and therefore it is the optimal path. In fact, if we compute the optimal path for 40 steps, the same result would be found.

Now we add some viscous damping to the system. In that case, the dynamics of the system is given by

\[ \ddot{x} = \begin{bmatrix} 0 & 1 \\ 0 & -10 \end{bmatrix} \dot{x} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u, \text{ where } \dot{z} = \begin{bmatrix} x \\ \dot{x} \end{bmatrix} \]
All the other conditions are the same as in the previous case. The results are presented in Fig. 7. The resulting path is the path that requires least control energy and, as one would expect in the case of the damped system, also the shortest possible path through the field with respect to distance.

Fig. 8 shows the computational costs of the used BFS (or equivalently FDP) algorithm, which is coded in C and has been running on a single PC with a Pentium IV 3.06 GHz CPU and 2.0 GB RAM. The figure shows the computational costs for a worst case scenario. The discrete set that has been searched does not contain obstacles, so each discrete point in the set has to be addressed. In this case, again the displacement of the system is restricted to one grid point each time-step. Furthermore the costs have been calculated for up to 50 prediction steps and several discrete velocity sets.

The low computation time of the algorithm makes it possible to use this control strategy in a real-time application. Further enlargement of the number of discrete velocities of $v_n$, the number of prediction steps or the number of nodes that can be addressed from each single node may contribute to a better performance of the controller. The computation time however, will rapidly increase. So we will have to consider a trade-off between the computation time and the other parameters such as the number of prediction steps. Since the BFS algorithm used in this approach is a forward search algorithm, we can use this approach even if the maximum allowable computation time is fixed, although the prediction steps may be changed. This is one advantage of this approach.

V. CONCLUSION

In this paper, an approximate version of the optimal control problem with non-convex state constraints has been addressed, applied to the obstacle avoidance problem. We first pointed out that the use of the standard continuous-time cost function does not give satisfactory results if non-convex state constraints are set, and then introduced a new cost function in which the intermediate target states are included. Next, for the problem with this cost function, where the non-convex state constraints are discretized with respect to time axis and state space, we derived an optimal continuous-time control in an explicit form, including the intermediate target states obtained as a solution of some discrete optimization problem, which can be efficiently solved via the Breadth First Search algorithm. Although the continuous-time trajectory obtained in this way is sub-optimal for the original (i.e., non-discretized) problem, it will be practically useful since it will be quite difficult to effectively solve the original problem. Furthermore, we presented some numerical experiments to prove the effectiveness of the proposed approach.

Future work will include the optimal way to discretize the space, to maximize the efficiency of the derived control strategy. Especially in the higher dimensional case, the computation time will rapidly increase, so the need to use more sophisticated sampling methods becomes evident. Although this paper mainly focuses on the obstacle avoidance problem, the approach proposed here can be extended to the more general class of the optimal control problem with non-convex state constraints, such as the control problem of hybrid systems. This will be another topic of interest that will be covered in future research.

Acknowledgements: The authors would like to thank Mr. Tatsuya Kanechika, Tokyo Institute of Technology, for his useful comments on the discrete optimization algorithm. This work was supported by TATEISI Science and Technology foundation.

VI. REFERENCES


