A Real-Time Traffic Control Scheme of Multiple AGV Systems for Collision Free Minimum Time Motion: A Routing Table Approach

Jung Hoon Lee, Beom Hee Lee, Senior Member, IEEE, and Myoung Hwan Choi, Member, IEEE

Abstract—A two-staged traffic control scheme, in which sets of candidate paths are prepared off-line prior to overall motion planning process, has been widely adopted for motion planning of mobile robots, but relatively little attention has been given to the application of the two-staged scheme to multiple automated guided vehicle systems (MAGVS’s). In this paper, a systematic two-staged traffic control scheme is presented to obtain collision-free minimum-time motions of AGV’s along loopless paths. The overall structure of the controller is divided into two tandem modules of off-line routing table generator (RTG) and an online traffic controller (OTC). First, an induced network model is established considering the configurational restrictions of guide-paths. With this model and a modified \( \text{shortest} \) path algorithm, RTG finds sets of \( k \)-candidate paths from each station nodes to all the other station nodes off-line and stores them in the form of routing tables. Each time a dispatch command for an AGV is issued, OTC utilizes these routing tables to generate a collision-free minimum-time motion along a loopless path. Real-time computation is guaranteed in that time-consuming graph searching process is executed off-line by RTG, and OTC looks for the minimum time motion among the \( k \)-candidate paths. The traffic control scheme proposed is suitable for practical application in centralized MAGVS with zone blocking technique.

I. INTRODUCTION

TRAFFIC control scheme for a multiple automated guided vehicle system (MAGVS) aims to solve the collision-free minimum-time motion planning problem, which is to determine the path (route) and trajectory of an AGV from a start node to a goal node when a dispatch command is issued [1], [2]. The traffic controller should find a path without any collision or deadlock, while the traveling time to its goal should be minimized.

In general, an AGV guidepath is considered as a network model which consists of nodes that represents stations and other control points, along with links that represents path segments connecting two control points [3]. The network model can be considered to be dynamically constrained, since if a node/link is scheduled to be occupied by an AGV for some duration, it will not be available to other AGV’s during that period of time [2]. The collision-free minimum-time motion planning problem can be converted to a routing problem under time-varying constraints on links and/or nodes of the network, which is described as a routing problem with time window constraints [1], [2], [4], [5].

Many approaches can be found that deals with the problem as a graph search problem. Broadbent et al. presented a collision-free minimum-time motion planning based on the centralized traffic control [6], where the potential collisions are detected by comparing node occupation times. Since Broadbent, the notion of node occupation times has been a prevailing idea in traffic control schemes and was adopted in many AGV systems [3], [7], [8]. Wing et al. [9] represents the guidepath and the travel of vehicles using symbolic temporal relationship. Their Quickest Journey algorithm is a combination of temporal...

In these works, the minimum-time path and trajectory for vehicle navigation is determined at the same time by an online traffic controller while avoiding any mutual collision between AGV’s. Indeed, the method using time window constraints can yield a flexible and optimal solution to the vehicle routing problem. However, it is difficult to obtain the path and trajectory simultaneously in real-time if the size of a network is large. Obtaining optimal motions by these schemes is known to be very time-consuming, even though the schemes possess polynomial time computation [1]. An alternative way to solve these difficulties is to apply two-staged traffic control scheme, i.e., to separate the task of finding the feasible candidate path sets and to perform this task off-line. For this scheme to be applicable to MAGVS, suitable algorithms both for path finding stage and for on-line traffic control stage are essential. However, relatively little attention has been given to applying two-staged scheme to minimum-time motion planning of MAGVS. Although the two-staged scheme has been widely adopted for motion planning of mobile robots, few previous works are available that give a thorough procedure to obtain the collision-free minimum-time motion plan using pre-established sets of feasible paths.

In this paper, a systematic two-staged traffic control scheme that separates the search process for the feasible candidate path sets from the overall minimum-time motion planning is presented. The scheme proposed in this paper is summarized as follows.

1) Since conventional guidepath network model itself is not appropriate for generation of candidate path sets, a converted network named induced network model is established from the conventional network model for a guidepath.

2) A modified $k$-shortest path algorithm is proposed to construct sets of multiple candidate paths that satisfies configurational restrictions of AGV guidepath. Each path set contains the shortest path and several substitute loopless candidate paths from a station node to all the other station nodes, and is computed off-line and stored in the form of routing tables. This stage needs to be performed only once for a given guidepath.

3) Each time a dispatch command for an AGV is issued, the on-line traffic controller (OTC) examines the routing tables and selects a path that results in the collision-free minimum-time trajectory. OTC yields minimum-time motion if the number of candidate paths contained in the routing table is sufficiently large. Real-time computation is guaranteed in that time-consuming graph searching processes are executed off-line, and OTC looks for the minimum-time motion among the finite candidate paths.

Fig. 1 describes the two-staged tandem structure.

This paper is organized as follows. Section II contains nomenclature and provides the basic system descriptions and assumptions on our MAGVS. In Section III, configurational restrictions in generating routing tables are discussed, and the procedure to establish induced network model is presented. Section IV presents a modified $k$-shortest path algorithm, which is employed in the routing table generator (RTG). Section V presents the procedure to construct OTC with its on-line motion planning algorithm. Two illustrative examples are given in Section VI. Section VII concludes the paper. Appendix A gives the proof for admissibility of the algorithm developed in Section IV. The computational complexities of the algorithms developed throughout Sections IV and V are analyzed in Appendix B.

II. DESCRIPTION OF MAGVS

Many practical plants adopt distributive traffic control, where the range sensors maintain a minimum headway between two adjacent vehicles [3]. These distributive control schemes cause a drop in efficiency of the system, especially when the complexity of the guidepath network is high. They are often not able to find substitute paths, and deadlock occurs more frequently with the increase of the number of AGV’s. Therefore the centralized traffic control scheme is preferable in a multiple-AGV system with complicated bidirectional guidepath. In real plants, the exact positions of AGV’s can be obtained only when they pass over pre-specified control points fixed on the guidepath, so the zone blocking technique which permits only one vehicle in a given path segment at a time is suitable for a multiple-AGV system under centralized traffic control [2], [3].
Fig. 2. Definition of starting angle and ending angle.

Fig. 3. An example: the reasonable second shortest “feasible” path is considered as the thirteenth shortest path by direct application of any available k-shortest path algorithm.

The zone blocking technique divides the guidepath into a set of many small path segments, and these segments are called links of the network. It should be noted that this notion of link in this paper is conceptual: the link in the network model need not correspond to a physical path partition that exists in a guidepath. One physical path partition of a guidepath may constitute many links on the network model.

The link from a node to one of its adjacent nodes is denoted by . For this link, the node is said to be adjacent from node whilst the node is adjacent to node . The link is assumed to be unique for each pair of adjacent nodes and . If the path segment between the node and is bidirectional, two links and are considered separately and the link is said to be the inverse link of .

Some general assumptions for practical MAGVS are adopted in this paper. Vehicles can start and stop only at nodes. The vehicle may stop temporarily at some node in the path to avoid collision. Spinturn of AGV on guidepath is assumed to be avoided. It is also assumed that a vehicle path cannot contain any loop, a partial path whose start node is the same as its goal node. The velocity profile for the vehicles is approximated to be trapezoidal, and the maximum speed for each profile is fixed at its maximum vehicle speed multiplied by the velocity parameter which is assigned to each link. This velocity parameter is introduced to decrease vehicle speed in curved links or other low velocity regions. Acceleration and deceleration of the AGV’s are assumed to be constants, and denoted by and . In the motion planning for a vehicle, the traffic controller cannot disrupt or reschedule any active travel schedule for other vehicles.

To evaluate the times required in traveling the paths, an index for the traveling time is needed. Define journey time as the traveling time required to move along a path . Note that the vehicle speed on each link is completely determined by the trapezoidal velocity profile which is again determined by the constants and . Therefore, we can estimate the journey time for a path from these variables and .

For link , its velocity profile has the maximum speed of . Therefore, when AGV is moving over link in constant maximum velocity, the time required to clear equals . Hence, the journey time for a path can be represented as

\[
\tau(P) = \sum_{(pq) \in P} (C_{pq \max})^{-1} \cdot d(p,q) + \sum_i \alpha_i + \sum_j \beta_j + \sum_r \eta_r
\]  

(1)

where

- \( \alpha_i \) additional time required for the \( i \)th acceleration;
- \( \beta_j \) additional time required for the \( j \)th deceleration;
- \( \eta_r \) additional time required for the \( r \)th temporary stay.

Journey time representation also enables us to estimate the entry time and the exit time associated with each link. Assume an AGV moves along a path and a link is in , then the entry time of is estimated as while the exit time is estimated as .

III. INDUCED NETWORK MODEL

Two-staged traffic control scheme requires construction of sets of multiple feasible candidate paths, which are called routing tables. A routing table for a start-goal node pair is a set of candidate paths which contains the shortest feasible path and second, third, , th shortest loopless feasible paths from station node to station node , while the number is chosen by the operator. Let the number of station nodes in a guidepath be equal to . Then, routing tables should be generated off-line.

In order to generate a routing table between each pair of start-goal nodes in a guidepath, a -shortest path algorithm is employed, which determines the first, second, , th shortest loopless paths for any pair of start-goal nodes. Many algorithms and their implementations have been proposed for the problem of finding shortest loopless paths. The basic technique on which these works are based...
Fig. 4. The induced network model for the guidepath shown in Fig. 3.

can be stated as partitioning the solution space [19], [13]. The outline of the technique is illustrated in subsection B of next section. However, the layout of the guidepath often restrict the direct application of these algorithms, since they may generate forbidden paths for which spinturn on guidepath is needed. A simple illustration for this case is shown in Fig. 3. For the given start node 1 and goal node 11, one would expect the path \( P = \{1, 3, 7, 10, 13, 14, 12, 11\} \) as the 2nd shortest path for travel of AGV's, while any straightforward application of a general \( k \)-shortest path algorithm would consider the path \( P' \) as the thirteenth shortest. The forbidden paths \( P_1, P_2, \ldots, P_{12} \) in Fig. 3 are the solutions of “unconstrained” \( k \)-shortest path problem, which neglects the configurational restriction on the successive linkage of two adjacent links.

This problem is solved by introducing the concept of starting angle and ending angle that describes the linkage between links contained in a path, and developing the corresponding constrained shortest path algorithm. Returning to Fig. 3, the inadmissibility of the forbidden paths arises from the improper linkages of two connected links in the paths. Hence, we can describe the case of inadmissibility through the concept of constraint on linkage between two connected links, or linkage constraint.

**Definition 1:** For a link \((p, q)\), the link \((q, r)\) is said to satisfy linkage constraint with \((p, q)\) if \( \angle(p, q) = \angle(q, r) \). In this case, \( r \) is said to be feasible from \( p \) via \( q \). Also, a network \( G(N, L) \) is said to be linkage-constrained if for some link \((p, q) \in L\), there exist a link \((q, r), r \in Adj(q) \) which does not satisfy linkage constraint with \((p, q)\).

Based on Definition 1, the concept of “feasibility” of a path can be introduced.

**Definition 2:** A path \( P \) consisting of more than two nodes is said to be feasible if
1) loopless and
2) for each subpath \( \{p, q, r\} \) of \( P, (q, r) \) satisfies linkage constraint with \((p, q)\).

Now, the shortest feasible path from a start node \( s \) to a goal node \( g \), denoted by \( \Pi_s \equiv \Pi_s[s, g] \), is defined as the shortest path among feasible paths from \( s \) to \( g \). Likewise, the \( k \)th shortest feasible path from \( s \) to \( g \), denoted by \( \Pi_{s,k} \equiv \Pi_{s,k}[s, g] \), is defined as the \( k \)th shortest path among feasible paths.

Using the notion of linkage constraint and feasibility, our constrained \( k \)-shortest loopless path problem can be stated as the problem of finding the first, second, \( \ldots, k \)th shortest feasible paths for a given starting node \( s \) and a goal node \( g \) in a linkage-constrained network. Now we are ready to define induced network model, which is a converted network from a given linkage-constrained network.

**Definition 3:** For a given linkage-constrained network \( G(N, L) \) and a start node \( s \), an induced network \( G_s(N_c, L_c) \) is generated such that

1) For each \((p, q) \in L\), a new element in \( N_c \) denoted by \((p, q)\), is defined. In this notation, \( p \) is called base node of \( p \).
2) For each pair of \( p \) and \( m \) in \( N_c \), a new element in \( L_c \) denoted by \((p, m)\), is defined if \( m \) is feasible from \( p \) via \( q \) and \( \eta \neq s \). Also, \( d(p, m) \equiv d(p) \). Also, \( G_s(N_c, L_c) \) is produced.
3) Finally, the root node of \( G_s(N_c, L_c) \), denoted as \( 0 \), is defined to be in \( N_c \). Also, for all \( s \in N_c \), define \((0, s) \) to be in \( L_c \), along with \( d(0, s) \). Then, \( G_s(N_c, L_c) \) is produced.

In order to explain the meaning of induced network model clearly, the notion of intension path is introduced below.

**Definition 4:** For a path \( \mathcal{P}[0, s, \eta] \equiv \{0, s, m, \ldots, p, \eta\} \) in \( G_s(N_c, L_c) \), the intension path of \( \mathcal{P} \) is defined as the path \( \{s, m, \ldots, p, \eta\} \) on \( G(N, L) \) constructed from the base nodes of each element in \( \mathcal{P} \).

The induced network model for the guidepath shown in Fig. 3. is illustrated in Fig. 4 with start node 1. Note that, the intension path of any path on \( G_s(N_c, L_c) \) is a feasible path on \( G(N, L) \), and inversely any feasible path on \( G(N, L) \) starting from \( s \) has a path on \( G_s(N_c, L_c) \) whose intension is this feasible path. Hence, a graph search algorithm that is performed on \( G_s(N_c, L_c) \) will deal only with feasible paths in \( G(N, L) \). In the next section, induced network model is utilized in Algorithm SFP to find shortest feasible paths. When
a search process on $G_s(N_c, L_c)$ is executed. Steps 1 and 2 of Definition 3 is performed once for a given guidepath model $G(N, L)$, whereas Step 3 of Definition 3 should be changed according to each start node.

IV. ROUTING TABLE GENERATION BY RTG

This section presents Shortest Feasible Path Algorithm (Algorithm SFP) and k-Shortest Feasible Path Algorithm (Algorithm KSFP), which finds the shortest and k-shortest feasible path(s) from a given start node $s$ to a goal node $g$, respectively. Algorithm SFP is employed to serve as a shortest feasible path algorithm that is required in the procedure of Algorithm KSFP.

A. Algorithm SFP

Algorithm SFP can be stated as a procedure to “grow” a tree $T_s(N_c^*, L_c^*)$ (which will be defined in Definition 5) using the original network $G(N, L)$ until the shortest feasible path from $s$ to $g$ is found.

Definition 5: For the start node $0s \in N_s$, a tree $T_s(N_c^*, L_c^*)$ on $G_s(N_c, L_c), N_c^* \subseteq N_c, L_c^* \subseteq L_c$, is defined such that the intension of each path $P[0s, p_n] = \{0s, \ldots, p_n\}$ in $T_s, \forall p_n \in N_c^*$ is the shortest feasible path $\Pi_s[p, n] = \{p, \ldots, p, n\}$ on $G(N, L)$.

Let a label $Dist(p_n)$ for each $p_n \in N_c^*$ denote the feasible path length from $0s$ to $p_n$. Also, define a priority queue $\mathcal{L}$, in which the leaf nodes in $T_s$ are sorted in their order of $Dist(\cdot)$ [20], [21]. Then, Algorithm SFP is described as follows.

[Algorithm SFP: Shortest Feasible Path Algorithm]

Step 0. [Initialization step: let $0s \in N_s$ be the only node included in $T_s$. Also set $Dist(0s) \equiv 0$. Let $0s \in \mathcal{L}$ as its only leaf node.]

- Let $N_c^* = \{0s\}$, Set $Dist(0s) = 0$.
- Let $\mathcal{L} = \{0s\}$.

Step 1. [Among the leaf nodes of $T_s$, pick one that has minimum $Dist(\cdot)$ value. If there is a tie, choose one arbitrarily. Let the leaf node picked be $p_n$. Exclude $p_n$ from $\mathcal{L}$]

- Choose $p_n \in \mathcal{L} \subset N_c^*$ such that

$$Dist(p_n) = \min_{n \epsilon \mathcal{L}} \{Dist(n)\}.$$  

- Exclude $p_n$ from $\mathcal{L}$, hence $p_n \in (N_c^* - \mathcal{L})$.

Step 2. [If $n = g$ then stop, and the path $P[0s, p_n]$ in $T_s$ is the shortest feasible path.]

Step 3. [Expand $p_n$. A child node of $p_n$, denoted by $n_{m}$, is generated by selecting $m$'s from $Adj(n)$. Not all $m$'s are selected even if $m$ is feasible from $p$ via $n$, but only those that meet the criteria below are chosen as the base node for the child node of $p_n$.]

- For $m \in Adj(n)$, let $n_{m} \in \mathcal{L} \subset N_c^*$ if:
  a) $m$ is feasible from $p$ via $n$, except the case $p_n = 0s$,
  b) $m$ is not a base node of the ancestor nodes of $p_n$ in $T_s$,
  c) $n_{m}$ is not repeated in $T_s$, i.e., $n_{m} \not\in N_c^*$ yet.
- For such $n_{m}$, let $(p_n, n_{m}) \in L_c^*$, and $Dist(n_{m}) := Dist(p_n) + d(p_n, n_{m})$.

Fig. 5. An illustration for finding shortest feasible path via Algorithm SFP.

Step 4. [Return to Step 1 if $\mathcal{L} \neq \phi$. If $\mathcal{L} = \phi$ then exit, since there exists no feasible path from $s$ to $g$.]

Note that Algorithm SFP works as a greedy algorithm [23], [13]. As $T_s$ “grows”, Algorithm SFP lays labels step by step on the new leaf nodes of $T_s$. Parenthetically, we can note here that Lemma 2 in Appendix A enables Algorithm SFP to employ the simple labeling operation $Dist(\cdot) := Dist(\cdot) + d(\cdot, n_{m})$, rather than the common formula $Dist(\cdot) := \min\{Dist(\cdot), Dist(p_n) + d(p_n, n_{m})\}$ which can easily be seen in many label-setting algorithms [13], [24]. In other words, $Dist(\cdot)$ is the only label employed in our Algorithm SFP, while many other label-setting methods use two kinds of labels—tentative and permanent [14], [15]. All labels $Dist(\cdot)$ are permanent, since the path from $0s$ to $n_{m}$ is unique in $T_s$. Since Algorithm SFP is a modification of an A$^*$ algorithm on $G_s(N_c, L_c)$, the procedure to check its admissibility is also focused on this modification, specifically on the criteria of Step 3. The admissibility is analyzed in Appendix A, and Appendix B contains the complexity analysis of Algorithm SFP.

An illustration for finding shortest feasible path via Algorithm SFP is depicted in Fig. 5. It might seem strange that no direct link exists in the model network from the start node $s$ to the goal node $g$, but it is part of the procedure of Algorithm KSFP to remove one or more links from the original network.

B. Finding k Shortest Feasible Paths

The following list summarizes notation introduced in describing the $k$-Shortest Feasible Paths Algorithm, or Algorithm KSFP.

- $\Psi$ Set of all feasible paths from node $s$ to node $g$.
- $n_s(j)$ $j$th node of $\Pi_s[s, g]$.
- $B[n] \triangleq \{(n, m) | m \in Adj(n)\}$.
- $S_c$ Set of candidate paths.
The concept of partitioning the solution space \[19], \[13] plays the key role in Algorithm KSFP. Provided that \(\Pi_1[s,g]\) exists, it can be found from Algorithm SFP. Now, let \(\pi_i\) denote the number of nodes included in \(\Pi_i[s,g]\), and define the feasible deviation path of \(\Pi_i[s,g]\) branching at \(n_i\) as the feasible path \(\{s, \cdots, n_i\} + \{n_i, \cdots, g\}\) where \(\{s, \cdots, n_i\}\) is the subpath of \(\Pi_i[s,g]\) and \(\{n_i, \cdots, g\}\) is not. Also, define a subset \(\Psi_j\) as the set of feasible deviation paths of \(\Pi_j[s,g]\) branching at \(n_j, j = 1, 2, \cdots, \pi_1 - 1\). Then, we can partition \(\Psi - \{\Pi_1\}\) into a family of subsets \(\Psi_1, \Psi_2, \cdots, \Psi_\pi - 1\). Notice that, from the definition of feasible deviation path, \(\Psi_1 \cup \Psi_2 \cup \cdots \cup \Psi_{\pi - 1}\) includes every feasible paths \(\Pi[s,g]\) except \(\Pi_1\). By the definition of \(\Pi_1[s,g]\), it is the shortest path among \(\Psi - \{\Pi_1\}\). Clearly, \(\Pi_2[s,g]\) lies in \(\Psi_1 \cup \Psi_2 \cup \cdots \cup \Psi_{\pi - 1}\), since \(\Psi_2, \Psi_3, \cdots, \Psi_{\pi - 1}\) is a partition of \(\Psi - \{\Pi_1\}\). Therefore, we can find \(\Pi_2[s,g]\) by selecting the shortest one among the shortest paths of \(\Psi_1, j = 1, 2, \cdots, \pi_1 - 1\) [13].

In similar way, we can obtain the \(k\)th shortest feasible path \(\Pi_k\) by partitioning \(\Psi - \{\Pi_1, \Pi_2, \cdots, \Pi_{k-1}\}\) into the sets of feasible deviation paths of \(\Pi_1, \Pi_2, \cdots, \Pi_{k-1}\), and picking up the shortest among the shortest paths computed for the sets. The complete procedure for Algorithm KSFP to find \(K\) different shortest feasible paths is described below.

[Algorithm KSFP: k-Shortest Feasible Paths Algorithm]

\**Step 0.** [Initialization step: find the 1st shortest feasible path with Algorithm SFP.]

- Set \(k = 1, S_c = \phi\).
- Find \(\Pi_1[s,g]\) with Algorithm SFP.

\**Step 1.** [Generation of feasible deviation paths: Find the shortest feasible deviation path from \(\Psi_j, j = 1, 2, \cdots, \pi_k\), which branches at \(j\)th node of the \(k\)th shortest feasible path. Put the paths found in \(S_c\).]

\*Loop A: For \(j\)th node of \(\Pi_k[s,g]\) \(j = 1, \cdots, \pi_k\) \{/ \\
  Define \(B^*\) as a subset of \(B[n]\). Let \((n_k(j), m) \in B'[n_k(j)]\) if \((n_k(j), m)\) is not repeated in \(\Pi_1[s,g], \Pi_2[s,g], \cdots, \Pi_k[s,g]\) and furthermore \(m\) is feasible from \(n_k(j - 1)\) via \(n_k(j)\) when \(j \neq 0\). \\
  \*Loop B: For each \((n_k(j), m) \in B'[n_k(j)]\) \{/ \\
    a) Set \(d(n_k(j)), * = \infty, \forall i = 1, \cdots, j - 1.\) \\
    b) Set \(d(n_k(j)), * = \infty\) except for \(d(n_k(j), m).\) \\
    c) Find \(\Pi_1[n_k(j), g]\) by applying Algorithm SFP. \\
    d) Return \(d(\ast, \ast, \ast)\) that are set to \(\infty\) in a) and b) to the former values. \\
    e) If \(\Pi_1[n_k(j), g]\) does not exist, return to a) for the next \((n_k(j), m)\). \\
    f) Put \(\{s, \cdots, n_k(j)\} + \Pi_1[n_k(j), g]\) in \(S_c\) (if it is not already in \(S_c\), where \(\{s, \cdots, n_k(j)\}\) is the subpath of \(\Pi_1[s,g]\)). \\
  \} \} \}

\*Step 2.** [Obtain \(\Pi_{k+1}[s,g]\): Find the shortest path in \(S_c\) and set the path as \(\Pi_{k+1}[s,g]\).]

- If \(S_c = \phi\) then exit, since \(\Pi_{k+1}, \cdots, \Pi_K\) distinguished from \(\Pi_1, \cdots, \Pi_k\) do not exist.
- Find the shortest one among paths in \(S_c\), and let it be \(\Pi_{k+1}[s,g]\).
- Remove \(\Pi_{k+1}[s,g]\) from \(S_c\).

---

Fig. 6. Data structure for LOT.

**Step 3.** [Check if \(k = K\).]

- If \(k = K\) then stop; otherwise, \(k = k + 1\) and return to Step 1.

Applying Algorithm KSFP iteratively, RTG constructs routing tables. Each routing table contains \(K\) loopless paths that have the same start and goal nodes, where the number \(K\) is defined by the operator. When applying Algorithm KSFP, the cost assigned to each link \((p, q)\) is replaced by \(C_{pq} \cdot d(p, q) + \Sigma_i \alpha_i + \Sigma_j \beta_j\) instead of \(d(p, q)\). The loopless paths in a routing table are sorted in the order of \(C_{pq} \cdot d(p, q) + \Sigma_i \alpha_i + \Sigma_j \beta_j\) and stored as data files.

V. MINIMUM-TIME MOTION PLANNING BY OTC

OTC works on the routing tables constructed by RTG, and finds collision-free minimum-time trajectory from the start node to the goal node. When a vehicle is dispatched, the link occupation schedule of the vehicle is stored in Link Occupation Table (LOT). Thus, LOT contains complete information on the trajectory of currently moving AGV’s. If a vehicle is required to move from its current node to a goal node, OTC plans collision-free trajectories along the \(K\) candidate paths in the routing table which contains \(K\)-shortest feasible paths from the current node to the goal node. In this process, LOT is exploited by OTC to ensure that two vehicles do not occupy a link at the same time. After the trajectory planning, the journey times required for moving along the paths are compared. Among those, the path with the smallest journey time is taken and OTC dispatches the AGV according to the motion planned.

A. Link Occupation Table

**Link occupation time** is the interval between the entry time and the exit time of a link. Link occupation table (LOT) is a table that stores link occupation times for every link, and used by OTC to determine a collision-free trajectory of a vehicle. The proposed data structure for each link is shown in Fig. 6.

In OTC, the travel of AGV’s are described as the time-occupying variations of links. In order to avoid collision against
other vehicles already in move, OTC should check LOT and plan the link occupation times of the vehicle not to overlap any link occupation times of other vehicles. Furthermore, to avoid collision at intersection nodes and also to guarantee safety, the neighbor links of physically occupied links have also to be considered occupied. In Fig. 7, an illustration for the calculation of link occupation time is described. If an AGV travels from node to its adjacent node during a time interval, the link and its inverse link and all the links and inverse links starting at or are all considered occupied during This protects vehicles from collision at intersection nodes. Notice that, in order to ensure minimum-time motion, it is enough to determine the length of each link associated with intersections to be the maximum length of vehicle size.

B. Trajectory Planning and the Minimum-Time Motion Planning

Using LOT, OTC can determine the collision-free minimum-time trajectory along a candidate path that is chosen from the routing table. Assume a candidate path was chosen, then OTC evaluates the link occupation schedule over , and check if the vehicle has any potential collisions by comparing its link occupation schedule with the predetermined schedules in LOT. It is considered that a potential collision occurs if its link occupation times overlaps any active occupation schedule by other vehicle. In this case, OTC calculates the temporary staying node and staying time to plan a modified trajectory for the candidate path.

1) Temporary Staying Node and Staying Time: The main purpose of the trajectory planning is to determine the temporary staying node and time. The temporary staying node and time are determined according to the types of collision. In the MAGVS proposed in our work, there exist four types of collisions [6]. A head-on collision can occur when two vehicles travel a link in opposite directions [Fig. 8(a)]. The rest three are cross collisions, which include the case at an intersection [Fig. 8(b)], the case if another vehicle is staying in the way [Fig. 8(c)], and the case if another vehicle is staying at the destination node [Fig. 8(d)]. Each type of collision determines different temporary staying nodes and times. The collision where a moving vehicle is hit from the rear by a faster moving vehicle [1] is not allowed, since the speed of the vehicle on the link is specified by the link velocity parameter .

In the case of a head-on collision, the trajectory is determined as in the illustration given in Fig. 9. Suppose that AGV #1 is a previously dispatched vehicle and AGV #2 is being considered for a new dispatch. AGV #1 has been scheduled along the path , and assume that the occupation time of AGV #2 for link overlaps the occupation time for (6, 5) that is previously scheduled by the travel of AGV #1. Then the temporary staying node and time for AGV #2 is calculated as follows.

[Algorithm TP: Trajectory Planning along a candidate path]

Step 1. [Check if any links in has potential collisions by referring to LOT. In our illustration, suppose a potential collision is detected on the link (6, 5).]
Step 2. [Get the index of AGV that occupies (6, 5). In our illustration, it is found that AGV #1 occupies the link.]

Step 3. [Take the inverse sequence of $\Pi_2$ and denote it by $inv(\Pi_2)$.]

$$inv(\Pi_2) \triangleq \{15, 14, 4, \ldots, 8, 12, 11\}.$$  

Step 4. [Take the start node $\sigma = 6$ of the link (6, 5) which bears the potential collision. Check the node sequence in $inv(\Pi_2)$ beginning from $\sigma$. The temporary staying node is determined as the first element $\xi \in inv(\Pi_2)$ after $\sigma$ such that $\xi \not\in \Pi_1$. In this illustration, $\xi = 12$.]

Step 5. [Pick $(\xi, \cdot)$ from $\Pi_2$, i.e., (12, 8) in this case. Denote $T_{12,8}$ as the expected entry time for (12, 8) by AGV #2. Among the exit times $O_{12,8}(i = 1, 2, \ldots, N)$ for link (12, 8), select $k$ such that $O_{12,8}$ is the smallest exit time which is greater than $T_{12,8}$.]

Step 6. [The temporary staying time $\eta_k$ is $O_{12,8} - T_{12,8}$. Modify the trajectory of AGV #2 to contain this temporary stay.]

Step 7. [Calculate $\alpha_{i \sigma}$ and $\beta_{i \sigma}$ for the temporary stay, and repeat Steps 1-6 until no potential collision is detected.] □

In the case of a collision at an intersection [Fig. 8(b)], the procedure for trajectory planning is the same as above, since the adjacent links for the intersection have been considered as occupied by the vehicle scheduled previously. In the case of collision with a vehicle staying in the way [Fig. 8(c)], the candidate path has $\eta_{\Pi} = \infty$ and this path is not available. Note that if a vehicle is staying at the goal node [Fig. 8(d)], no solution exists in the motion planning problem.

2) Collision-Free Minimum-Time Motion Planning Algorithm: Paths in each routing table are sorted in the order of their journey time assuming $\eta_{\Pi} = 0, \forall \sigma$. Hence, when start and goal nodes are given and starting time is determined, OTC starts examining the paths in the corresponding routing table, beginning with $\Pi_1$ which has the smallest $\tau(\Pi_1)$ in the routing table. Suppose that a start-goal node pair $s/g$ is given, and $RT_{s/g} \triangleq \{\Pi_1, \Pi_2, \ldots, \Pi_K\}$ denote the routing table for $s/g$ pair. Also assume that the collision-free trajectory for each path $\Pi_i, i = 1, \ldots, K$ have been evaluated so that the minimum journey time $\tau(\Pi_i)$ for $\Pi_i$ has been found. Then, obviously the collision-free minimum-time trajectory in $RT_{s/g}$ is the trajectory that has the smallest $\tau(\Pi_i)$. Hence, if $K$ is sufficiently large, the collision-free minimum-time loopless trajectory from $s$ to $g$ is the trajectory in $RT_{s/g}$ that has the smallest $\tau(\Pi_k)$, since the routing table $RT_{s/g}$ contains all possible paths from the start node to the goal node.

The collision-free minimum-time motion planning procedure is summarized below.

[Algorithm MP: Overall Motion Planning]

Step 1. [Initialization] $i := 1$.

Step 2. [Plan the trajectory along the path $\pi_i$ by Algorithm TP.]

Step 3. [If no potential collision is detected for $\Pi_i$ (i.e., $\Sigma_i \eta_{\Pi_i} = 0$), the collision-free minimum-time path lies among $\Pi_1$, or $\Pi_2, \ldots$, or $\Pi_K$. The path $\Pi_i(1 \leq j \leq i)$ that has the smallest $\tau(\Pi_i)$ among $\tau(\Pi_1), \tau(\Pi_2), \ldots, \tau(\Pi_K)$ is selected. Motion planning is finished with the trajectory determined in Step 2.]

Step 4. [If $\Sigma_i \eta_{\Pi_i} > 0$ and $i < K$ for $\Pi_i$, return to Step 2 with $i := i + 1$.]

Step 5. [If $\Sigma_i \eta_{\Pi_i} > 0$ and $i = K$ for $\Pi_i$, the path $\Pi_i(1 \leq j \leq K)$ that has the smallest $\tau(\Pi_i)$ among $\Pi_1$, or $\Pi_2, \ldots$, or $\Pi_K$ is selected. Motion planning is finished with the trajectory determined in Step 2.]

Since a routing table has $K$ paths, the motion planning operation has maximum $K$ iterations. Finite number of iterations enables us to achieve reliable real-time motion planning. The solution generates the collision-free minimum-time path if the user-defined number $K$ is sufficiently large, whilst the number $K$ is determined to be suitable for the guideway configuration. On the other hand, no deadlock occurs in the traffic control system, since the feasibility of the travel schedule is checked and guaranteed in the stage of trajectory planning.

VI. AN EXAMPLE

A. Generation of Routing Table

For an illustrative example for Algorithm KSFP, a model guideway has been designed and the generation of routing table is tested. Fig. 10 shows the designed guideway and the results obtained by applying Algorithm KSFP on the guideway with $K = 5$. Velocity parameters $C_{ij}$ are given 1 for straight links and 0.5 for curved links.
B. Real-Time Traffic Control on OTC

A simulation has been run on an IBM PC 486 DX-66 environment. The model guidepath tested in the simulation is shown in Fig. 11. The specifications were as follows:

1) Number of nodes \([N]\): 186.
2) Number of links \([L]\): 466 (233 x 2, bidirectional links).
3) Number of vehicles \(V\): 3.
4) Maximum vehicle speed \(V_{\text{max}}\): 1.0 m/s (for both directions).
5) Acceleration \(a_{\text{acc}}\), deceleration \(a_{\text{dec}}\): 1.0 m/s².

Table I contains the amounts of the computation time required for on-line motion planning in various cases. The motion planning was for AGV #3, and was performed after the scheduling of AGV #1 and #2. The path of AGV #1 was from STATION 4 to STATION 3, and AGV #2 from STATION 9 to STATION 5. AGV #3 had to be scheduled to travel to its goal node without any collision against AGV #1 or #2. The amount of computation time varied with the path of AGV #3 and with the number of potential collision it met. Various start and goal nodes of AGV #3 were tested.

Table I shows that the computation time on OTC for motion planning is under 0.3 s even in the worst case. The computing times are acceptable for real-time applications, since they are well within the range of the intervals of vehicle requests in most practical MAGVS’s.

<table>
<thead>
<tr>
<th>Start node</th>
<th>Goal node</th>
<th>Computation time</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATION #1</td>
<td>STATION #6</td>
<td>0.17 sec</td>
</tr>
<tr>
<td>STATION #1</td>
<td>STATION #8</td>
<td>0.17 sec</td>
</tr>
<tr>
<td>STATION #1</td>
<td>STATION #10</td>
<td>0.22 sec</td>
</tr>
<tr>
<td>STATION #1</td>
<td>STATION #11</td>
<td>0.20 sec</td>
</tr>
<tr>
<td>STATION #6</td>
<td>STATION #1</td>
<td>0.06 sec</td>
</tr>
<tr>
<td>STATION #8</td>
<td>STATION #1</td>
<td>0.08 sec</td>
</tr>
<tr>
<td>STATION #10</td>
<td>STATION #1</td>
<td>0.08 sec</td>
</tr>
<tr>
<td>STATION #11</td>
<td>STATION #1</td>
<td>0.06 sec</td>
</tr>
<tr>
<td>STATION #3</td>
<td>STATION #4</td>
<td>0.11 sec</td>
</tr>
<tr>
<td>STATION #5</td>
<td>STATION #11</td>
<td>0.08 sec</td>
</tr>
<tr>
<td>STATION #10</td>
<td>STATION #2</td>
<td>0.08 sec</td>
</tr>
</tbody>
</table>

VII. Conclusions

A two-staged traffic control scheme for MAGVS has been developed in which minimum-time motion along loopless paths is guaranteed. Advantages of applying two-staged scheme comes mainly from the fact that real-time traffic control is possible even for large-scaled MAGVS’s. Although two-staged scheme itself has been widely adopted for motion planning of mobile robots, relatively little attention has been given to applying this scheme to minimum-time motion planning of MAGVS. In this paper, the systematic procedure to obtain collision-free minimum-time motions for MAGVS is proposed and verified. The proposed traffic control system consist of an off-line RTG and an OTC. The off-line RTG took complete charge of path search process, to reduce the burden of on-line computation by OTC.

First, since conventional network model itself is not appropriate for generation of candidate path sets due to configurational restriction of AGV guidepaths, this restriction was represented as linkage constraint, and induced network model was proposed as a converted network for linkage-constrained network. Next, a modified shortest and \(k\)-shortest
path algorithm was developed to construct sets of $K$ feasible candidate paths that are stored in routing tables. The off-line stage of generating routing tables needs to be performed only once for a given network paths. Finally, based on the routing tables, a motion planning algorithm that guarantees a real-time solution was presented. Algorithm TP with MP gives a fast procedure to obtain the collision-free minimum-time motion by means of the pre-established routing tables. The paths planned through this method are restricted to be loopless, and the problem of obtaining motions including loop paths are left for future research.

Through an example, the real-time operation of the on-line traffic control was demonstrated. This operation can be performed in real-time, even in the case that the guideway network has considerably large number of nodes. The traffic control scheme presented is appropriate for a centralized MAGVS with zone blocking technique, and is suitable for practical applications.

**APPENDIX A**

**ADMISSIBILITY OF ALGORITHM SFP**

In order to verify the admissibility of Algorithm SFP, the following lemmas are required.

**Lemma 1:** If $P[s,n] = \{s, \ldots, p, n\}$ is loopless intension of $P_{1}[0s,p,n]$, then $P[s,n]$ is the shortest feasible path on $G(N,L)$ from $s$ to $n$ among the paths ending with $\{p, n\}$.

**Proof:** Obviously $P[s,n]$ is feasible, hence it is sufficient to show that $P[s,n]$ is the shortest among feasible ones that end with $\{p, n\}$. Let $P_{1}[0s,p,n]$ be represented as $P_{1}[0s, q, n] = \{0s, s, m, \ldots, s, p, n\}$, then $P[s,n] = \{s, m, \ldots, s, p, n\}$. Suppose there exists a feasible path $P'[s,n] = \{s, v, \ldots, w, p, n\}$ on $G(N,L)$. Since any arbitrary subpath $\{s, \ldots, w\}$ of $P'$ is feasible, the path $\{0s, s, v, \ldots, w, p, n\}$ is defined on $G_{s}(N_{c}, L_{c})$ and is different from $P_{1}$. Now, by the definition of $P_{1}[0s,p,n]$, $d(\{0s, s, m, \ldots, s, p, n\}) \leq d(\{0s, s, v, \ldots, w, p, n\})$. Therefore, $d(P) = d(\{s, m, \ldots, s, p, n\}) \leq d(\{s, v, \ldots, w, p, n\}) = d(P')$. Since $P'$ is chosen arbitrarily from the feasible paths ending with $\{p, n\}$, the proof is completed. □

**Lemma 2:** For $\{p, n\}$ obtained in each iteration, let $\eta_{m} \in N_{c}$. Then, the in-degree [13] of $\eta_{m}$, the number of links adjacent to $\eta_{m}$, equals 1.

**Proof:** Suppose the in-degree of $\eta_{m}$ is more than 1. Then, there exist $u, v, \eta_{m} \in N_{c}$ such that $\{u, v, \eta_{m}\}$ and $\{v, \eta_{m}\} \in L_{c}$. Without loss of generality, assume $u \eta_{m}$ has been expanded before $v \eta_{m}$. Since $u \eta_{m}$ is feasible from $u$ via $\eta_{m}$, and hence $u \eta_{m}$ is in $T_{s}$. Obviously in later steps, $v \eta_{m}$ cannot have $\eta_{m}$ as its child node by criterion iii) in Step 3, so $\{v, \eta_{m}\} \in L_{c}$. This contradicts the assumption. □

**Lemma 3:** For $N_{c}^{t}$ and $L$ obtained in each iteration, $\eta_{m} \in (N_{c}^{t} - L)$ implies that

$$D_{\eta_{m}}(p_{n}) = d(P_{1}[0s,p,n])$$  (2)

where $P_{1}[0s,p,n]$ indicates the shortest path on $G_{s}(N_{c}, L_{c})$ from $0s$ to $p_{n}$.

**Proof:** The proof is inductive. Clearly, (2) holds after the first iteration, so the base for our induction is thus obtained. Next, assume that (2) holds just before Step 1 is performed. Let the leaf node picked in Step 1 be $\eta_{m} \in L$, which has the smallest $D_{\eta_{m}}(p_{n})$. After Step 3, $\eta_{m} \in (N_{c}^{t} - L)$, and $D_{\eta_{m}}(p_{n}) = d(P_{1}[0s,p,n])$ in which $p_{n}$ is the unique parent node of $\eta_{m}$ (by Lemma 2). Also, all the ancestor nodes of $\eta_{m}$ belong to $(N_{c}^{t} - L)$, Hence, from the induction hypothesis (2) $D_{\eta_{m}}(p_{n}) = d(P_{1}[0s,p,n])$. Now, if we can show that $P_{1}[0s, \eta_{m}] = P_{1}[0s,p_{n}] + \{p_{n}, \eta_{m}\}$, then the proof is completed.

Suppose there exist $q \neq p$ and a path $\{0s, \ldots, s_{n}, \eta_{m}\}$ including $\eta_{m}$, such that $m$ is feasible from $q$ via $n$ and

$$d(\{0s, \ldots, s_{n}, \eta_{m}\}) < d(\{0s, \ldots, p_{n}, \eta_{m}\}).$$  (3)

**Case i)** Suppose $\eta_{m} \in (N_{c}^{t} - L)$. If it is the case, $\eta_{m}$ is a child node of $\eta_{m}$ since $\eta_{m}$ has been already expanded. Hence, by Lemma 2, $p_{n}$ cannot be a parent node of $\eta_{m}$, which is contradictory.

**Case ii)** Suppose $\eta_{m} \in L$. Provided that $\eta_{m} \in L$, Lemma 2 tells us that there exists a unique parent node of $\eta_{m}$, $u \in (N_{c}^{t} - L)$, such that $u$ is feasible from $u$ via $q$. Then, from the induction hypothesis (2)

$$D_{\eta_{m}}(p_{n}) = D_{\eta_{m}}(q) + d(q, \eta_{m})$$

$$= d(P_{1}[0s, u], q) + d(q, \eta_{m})$$

$$= d(0s, \ldots, u, q, \eta_{m}).$$

Hence, from the fact that $D_{\eta_{m}}(p_{n})$ is the smallest

$$d(0s, \ldots, u, v, \eta_{m})$$

$$\equiv d(0s, \ldots, u, v, \eta_{m})$$

$$d(0s, \ldots, u, v, \eta_{m})$$

$$\geq D_{\eta_{m}}(p_{n}) = d(0s, \ldots, u, q, \eta_{m})$$

which contradicts (3).

**Case iii)** Suppose $\eta_{m} \notin N_{c}^{t}$. In this case, we can find a subpath $\{u_{v}, u_{w}\}$ from $0s, \ldots, u_{w}, \eta_{m}$ such that $u_{v} \in N_{c}^{t}$, $u_{w} \notin N_{c}^{t}$ and $u$ is feasible from $u$ via $v$, since $0s \in N_{c}^{t}$ and $\eta_{m} \notin N_{c}^{t}$. Suppose $u_{v} \notin L \subset N_{c}^{t}$, Then, as we can see in case ii)

$$d(0s, \ldots, u_{v}, \eta_{m})$$

$$\geq d(0s, \ldots, u_{v}, \eta_{m})$$

$$\geq D_{\eta_{m}}(p_{n})$$

$$\equiv d(0s, \ldots, u_{v}, \eta_{m})$$

which contradicts (3). Hence, $u_{v} \notin L$. However, this cannot be the case either, since the existence of subpath $\{u_{v}, u_{w}\}$ requires that $u_{v} \notin N_{c}^{t}$ is a child node of $u_{w} \in (N_{c}^{t} - L)$.

Therefore, $\eta_{m}$ that meets (3) does not exist, and it is verified that the induction remains true for every iterations. □

**Lemma 4:** When Algorithm SFP is terminated be reaching

$$i_{g}, \eta \in N$$

$$d(P[0s, i_{g}]) \leq d(P[0s, \eta_{j}])$$

for any $\eta \in N_{c}, \eta \in N$. 


Proof: \( \text{Dist}(i^g) \leq \text{Dist}(p\eta) \) for any \( p\eta \in \mathcal{L} \), since \( i^g \) is the node with \( \min \{ \text{Dist}(\omega) \} \). Suppose \( i^g \notin N^*_L \). Then, as in the case iii) in the proof of Lemma 3

\[
\text{Dist}(i^g) \equiv d([0^v, s, \cdots, u^v, w, \cdots, i^g]) > d([0^v, s, \cdots, u^v]) \equiv \text{Dist}(u^v) \geq \text{Dist}(i^g) \quad (4)
\]

where \( u^v \in N^*_L, u^v \notin N^*_L \). Otherwise, if \( j^g \in N^*_L \), obviously \( j^g \in \mathcal{L} \), and by (4) \( d(\mathcal{P}[0^v, j^g]) \leq d(\mathcal{P}[0^v, i^g]) \).

Now, we come to the proof for the admissibility of Algorithm SFP.

Observation 1: Algorithm SFP is admissible, i.e., when Algorithm SFP is terminated by reaching \( i^g, i \in N \), the intension of the path \( \mathcal{P}[0^v, i^g] \) in \( \mathcal{T}_s \) from \( 0^v \) to \( i^g \) equals \( \Pi_{i^g} \), the shortest feasible path on \( G(N, L) \) from \( s \) to \( g \).

Proof for Observation 1: First, notice that \( i^g \in (N^*_L - \mathcal{L}) \subset N^*_L \) when Algorithm SFP is terminated at Step 2. By Lemma 2, it is guaranteed that \( \mathcal{P}[0^v, i^g] \) is the unique path from \( 0^v \) to \( i^g \) in \( \mathcal{T}_s \). Hence, by the definition of \( \text{Dist}(\cdot) \), \( d(\mathcal{P}[0^v, i^g]) = \text{Dist}(i^g) \). From the uniqueness alone with Lemma 3, it follows that

\[
\mathcal{P}[0^v, i^g] = \mathcal{P}[0^v, i^g]. \quad (5)
\]

On the other hand, by criterion b) in Step 3, the intension of \( \mathcal{P}[0^v, i^g] \) in \( \mathcal{T}_s \) is loopless, and by criterion a), it is also feasible. Therefore, by Lemma 1, the intension of \( \mathcal{P}[0^v, i^g] \) equals the shortest feasible path from \( s \) to \( g \) among the paths which ends with \( (i, g) \). Lemma 4 verifies that it is the shortest feasible path from \( s \) to \( g \) on \( G(N, L) \). \( \square \)

APPENDIX B

Complexity Analysis

A. Complexity Analysis for Algorithm SFP and Algorithm KSFP

First, let us check the computational complexity for Algorithm SFP. Denote \( \nu = |N| \) as the number of nodes in the network. Step 1 itself requires \( O(\nu) \) computation in choosing the node with smallest \( \text{Dist}(\cdot) \). The criterion ii) in Step 3 requires all the ancestor nodes in \( \mathcal{T}_s \) to be examined, so it takes a time in \( O(\nu) \) in the worst case. The criterion iii) also requires at most \( |N^*_L| + 2 \) times, since each node it picks from \( N^*_L - \{s\} \) cannot be selected twice. Therefore, Algorithm SFP takes a time in \( O(\nu^2) \).

In addition, it is obvious that the number of nodes generated in \( \mathcal{T}_s \) is bounded below \( M \cdot |N| + 1 \), since \( |N^*_L| \leq |L| + 1 \leq M \cdot |N| + 1 \), where \( M = \max_{p \in N} \text{Adj}(p) \leq (|N| - 1) \). In usual guidepath layouts, \( M \) can be considered as a constant no more than 10. This implies \( |N^*_L| \leq 10 \cdot |N| \), which will guarantee the practicality of the algorithm. Hence, the memory requirement for the implementation of Algorithm SFP is acceptably small.

Next, let’s check the complexity of Algorithm KSFP. Consider Step 1 first. In Loop A, checking whether \( (\eta_k(j), m) \) is in \( \Pi_k \), demands \( (\pi_1 + \cdots + \pi_k) \) times of comparisons, where \( (\pi_1 + \cdots + \pi_k) \leq (\nu + \cdots + \nu) \leq K \cdot \nu \leq \nu^2 \) since \( K \ll \nu \) in general.

Procedure a) in Loop A requires \( M \pi_k \leq MV \in O(\nu) \) times of computations, while c) does \( O(\nu^2) \). Since Loop B is repeated at most \( M \nu \) times, it can be said that Loop B takes a time in \( O(\nu^2) \). Meanwhile, Loop A is iterated \( \pi_k \leq \nu \) times, hence it requires computation time in \( O(\nu^3/2) \). As a whole, Steps 1–3 is repeated \( K \) times, therefore Algorithm KSFP takes a time in \( O(K\nu^2) \).

B. Computational Complexity of Algorithm MP

The traffic control system proposed is a two-staged system composed of RTG and OTC module. Since the time-consuming operations for path search procedure are performed by RTG in off-line, a relatively small amount of operations are performed by OTC.

The computational complexity of Algorithm MP can be derived by assessing the upperbound for the complexity in worst-case condition. Let an MAGVS has \( V \) vehicles, and each routing table has \( K \) paths. Also, let \( S \) represent the maximum number of nodes in a path. Step 2 is the process of Algorithm TP, and it is easily checked that it takes a time in \( O(VS^2) \). Steps 2–5 loop is iterated \( K \) times in the worst case. Hence the upperbound for the complexity is assessed to be \( O(KVS^2) \).

REFERENCES


Jung Hoon Lee received the B.S. and M.S. degrees in control and instrumentation engineering from Seoul National University, Seoul, Korea, in 1992 and 1994, respectively. He is currently pursuing the Ph.D. degree at the School of Electrical Engineering at Seoul National University. His research interests include traffic control of multiple mobile robots, precise positioning of wheeled mobile robots, and control of flexible joint manipulators.

Beom Hee Lee (S’84–M’85–SM’92) received the B.S. and M.S. degrees in electronics engineering from Seoul National University, Seoul, Korea, in 1978 and 1980, respectively, and the Ph.D. degree in computer, information, and control engineering from the University of Michigan, Ann Arbor, in 1985.

From 1978 to 1980, he was a Departmental Assistant in the Department of Electronics Engineering, Seoul National University, and from 1980 to 1981 he was an Instructor at Chung-Ang University, Seoul, Korea. From 1985 to 1987 he was with the School of Electrical Engineering, Purdue University, West Lafayette, IN, as an Assistant Professor. He joined Seoul National University in 1987, where he is currently a Professor in the School of Electrical Engineering. Since 1995, he has been Chairing the Seoul Metropolitan Business Incubator. His research interests include path/trajectory planning, kinematics and dynamics, obstacle avoidance of mechanical manipulators, multirobot operations, computer integrated manufacturing, artificial intelligence, and control system analysis and design.

Myoung Hwan Choi (S’89–M’93) received the B.S., M.S., and Ph.D. degrees in control and instrumentation engineering from Seoul National University, Seoul, Korea, in 1986, 1988, and 1992, respectively. He was with the Automation and Systems Research Institute (ASRI), Seoul National University, in 1992, and the Oak Ridge National Laboratory, Oak Ridge, TN, from 1995 to 1996. He is currently an Assistant Professor in the Department of Control and Instrumentation Engineering, Kangwon National University, Chunchon, Korea. His research interests include industrial robot controllers, multirobot coordination, robot force control, robot teaching, mobile robots, and manufacturing automation.