2014 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS 2014)
September 14-18, 2014, Chicago, IL, USA

Cameraman Robot: Dynamic Trajectory Tracking with Final Time Constraint using State-time Space Stochastic Approach

Igi Ardiyanto and Jun Miura

Abstract— **This paper describes an approach to realize a cameraman robot. Here a robot is given a task to follow a certain trajectory for taking the video of an actor in a designated time. The trajectory is relative to the actor, so that the robot has to take into account the actor's movement. We map the given trajectory to a new one in the state-time space based on the prediction of the actor's pose. We then build a trajectory tracking system using a 3D time-space wavefront potential considering the robot kinematic, trajectory cost, obstacles, and visibility constraints. This potential is then used for generating the robot motion control by using a modified random tree search algorithm with the control law. Simulation results show the effectiveness and feasibility of our approach.**

Fig. 1: An illustration of *dolly track*[1]

I. Introduction

In the process of making a movie or a TV program, we often see a cameraman takes a video from a moving base mounted on a track, like the Fig. 1. This track structure is called *dolly track*. There is also so-called *pedestal*, a wheeled-base mounting for the camera [1]. Both are used for creating a smooth and stable camera movement.

Let us take an example where the director of the movie or TV program needs to plan how the camera should move to take the video. In an advertisement program, the director wants to slowly take the video of a standing kid from his right-side to the left-side in 10 seconds showing him eating an ice cream product. A dolly track then can be configured, for example in front of the kid forming a curve, and then the cameraman should pull the camera on top of the track in the planned time and pose.

The scenario above is then changed, the company wants the same setting while the kid walks around. The director has some options; the camera is handled directly by a cameraman with a degraded camera stability in return; or, changes the dolly track by considering the kid movement.

We are interested in the second case, where the dolly track is used. Obviously, this method is not handy because there are some efforts for replacing the track when the scenario is changed. For example using the case above, changing the curve path of the camera to a straight line means we need completely change the dolly track. Moreover, some obstacles such as studio pillars may block the path of the dolly track at the set or studio. We offer the usage of a robotic system as solution of this problem.

Basically we replace the dolly track with a mobile robot system. The robot is then given the scenario as above, where the planned movement, time, and the camera pose can be easily changed according to the director, for example by using a sketch. This scenario resembles a trajectory tracking problem in the robotic field ([3], [6]), with additional issues, such as *final time constraint*, *dynamic trajectory*, and *camera visibility*. Several popular methods for solving the trajectory tracking, such as LQR [2] and MPC ([3], [4], [5], and [6]) do not fully cover our cameraman robot problem, as we will explain later.

Here we develop a new approach to solve those problems. We deal with a dynamic actor by bringing the relative trajectory into the state-time space, considering the actor movement. The state-time space is ranged from the current time to the final time constraint. Several cost maps are then built for handling the obstacles, camera visibility, and trajectory error. These maps are then used for building a wavefront potential, which will guide a modified randomized tree search with the control law. The first control of the best tree is then chosen, and the process is repeated until the robot reach the end of the trajectory.

Our contributions are the construction and formalization of the cameraman robot problem as a dynamic trajectory tracking (*i.e.* trajectory tracking with a dynamic reference) with the final time constraint, and the usage of a stochastic approach for solving it, by utilizing wavefront potential and random tree search which consider the obstacles and camera visibility.

The rest of our paper is organized as follows. In section II we construct the definition of the cameraman robot problem. We then build up a comprehensive strategy for solving the cameraman robot problem in section III. We then verify the experiment results in section IV. The rest is the conclusion and possible future works.

I. Ardiyanto and J. Miura are with Department of Computer Science and Engineering, Toyohashi University of Technology, Hibarigaoka 1-1, Tenpaku-cho, Toyohashi, Aichi, 441-8580, Japan {iardiyanto,jun}@aisl.cs.tut.ac.jp

[1]The action in the figure is done by the author and just intended for giving an illustration only, not an act of a professional.

II. Dynamic Trajectory Tracking with Final Time Constraint

In this section, we construct and transform the cameraman robot problem into a dynamic trajectory tracking problem with the final time constraint.

A. Problem Statement

1) Definition of Cameraman Robot Task: Given a scene of a movie where the cameraman person is required to follow a certain trajectory $\mathcal{P}$ relative to the actor $\mathcal{A}$ in a determined time T_f for taking the film, the actor is not necessarily motionless (*i.e.* can take any movement). The trajectory $\mathcal{P}$ can be at any form up to the movie director's desire, *e.g.* a straight line, a circular, or even a curve relative to the actor.

Let us imagine that the cameraman person is then replaced by an omnidirectional robot $\mathcal{R}$ equipped by a static camera. Here the robot needs to imitate the cameraman person task, *e.g.* to follow the trajectory $\mathcal{P}$ while keeping the camera orientation towards the actor $\mathcal{A}$. To simplify the problem, we assume the movement of $\mathcal{A}$ follows a constant velocity motion model, as will be described in subsection II-B. The determined time T_f is then defined as the interval time from which the robot has to reach the end of the trajectory $\mathcal{P}$, and from now, it is called as *final time constraint*.

2) Comparison with Established Method: When the actor $\mathcal{A}$ is motionless and the final time constraint T_f is not considered, the problem above becomes a standard trajectory tracking problem. The trajectory tracking problem (without the final time constraint) has been derived and investigated by many researchers using several methods, for example *Linear Quadratic Regulator* (LQR) [2], *Model Predictive Control* (MPC) ([3], [4], [5], and [6]), and *Sliding Mode Control* (SMC) [9].

We now take a look on the MPC which is widely used for solving the trajectory tracking problem. The basic idea is to search for a set of control which minimizes the error cost to the desired trajectory in a fixed time horizon, take the first control, and then repeat.

Let $\boldsymbol{q} \in \mathbb{Q}$ be the state of the robot $\mathcal{R}$, $\boldsymbol{u} \in \mathbb{U}$ be the robot control, and $\dot{\boldsymbol{q}}$ be the dynamics of $\mathcal{R}$ given by

$$\dot{\boldsymbol{q}} = f(\boldsymbol{q}, \boldsymbol{u}), \tag{1}$$

where $\mathbb{Q}$ and $\mathbb{U}$ respectively represent the state space and the control space of $\mathcal{R}$. Let $\tilde{\boldsymbol{u}} : [t_0, T_H] \rightarrow \mathbb{U}$ be the control sequences from the current time t_0 to T_H. The MPC is then generally defined as a minimization of the objective function J to find the optimal control sequence $\tilde{\boldsymbol{u}}^* \leftarrow \tilde{\boldsymbol{u}}$, given by

$$\tilde{\boldsymbol{u}}^* = \min_{\tilde{\boldsymbol{u}}} J \simeq g(\boldsymbol{q}, \boldsymbol{u}, T_H), \tag{2}$$

subject to

$$\begin{aligned} \dot{\boldsymbol{q}} &= f(\boldsymbol{q}, \boldsymbol{u}), \\ \boldsymbol{q_{min}} \leq \boldsymbol{q} &\leq \boldsymbol{q_{max}}, \\ \boldsymbol{u_{min}} \leq \boldsymbol{u} &\leq \boldsymbol{u_{max}}, \end{aligned} \tag{3}$$

where the second and third inequality term of eq. 3 respectively represent the state (*e.g.* boundary/collision) and control (*e.g.* velocity/acceleration) constraints.

For the trajectory tracking problem, the objective function $g(.)$ is associated with the error between the state and the reference. Given a priori known relative trajectory (as mentioned in the previous subsection) $\mathcal{P} := \tilde{\boldsymbol{q}}_r \subset \mathbb{Q}$,

$$\tilde{\boldsymbol{q}}_r = \{\boldsymbol{q_{r}}_0, \boldsymbol{q_{r}}_1, \ldots, \boldsymbol{q_{r}}_f\} \in \mathcal{P}, \tag{4}$$

where $\tilde{\boldsymbol{q}}_r$ denote the state reference sequences, $\boldsymbol{q_{r}}_0$ and $\boldsymbol{q_{r}}_f$ denote the initial and final state of the trajectory. Subsequently, eq. 2 becomes

$$\begin{aligned} \tilde{\boldsymbol{u}}_{\boldsymbol{k}}^* = \min_{\tilde{\boldsymbol{u}}_{\boldsymbol{k}}} J_k \simeq m(\boldsymbol{q}_{T_H}, \boldsymbol{u}_{T_H}) + \sum_{k=t_0}^{t_0+T_H} \ell(\boldsymbol{q_e}_{\boldsymbol{k}}, \boldsymbol{u_k}), \\ \boldsymbol{q_e}_{\boldsymbol{k}} = \boldsymbol{q_k} - \boldsymbol{q_r}_{\boldsymbol{k}}, \end{aligned} \tag{5}$$

where $m(.)$ and $\ell(.)$ respectively denote the terminal and integrand cost, q_{e_k} is the state error, and T_H is the time horizon. In the case of linear/linearized system, some researchers (*e.g.* [5], [7], and [8]) turn function $\ell(.)$ into a *Quadratic Programming* (QP) problem, given by

$$\ell(\boldsymbol{q_e}_{\boldsymbol{k}}, \boldsymbol{u_k}) = \boldsymbol{q_e}_{\boldsymbol{k}}^T Q \boldsymbol{q_e}_{\boldsymbol{k}} + \boldsymbol{u}_{\boldsymbol{k}}^T R \boldsymbol{u_k}, \tag{6}$$

where Q and R denote positive semidefinite matrices for weighting respectively the state and control.

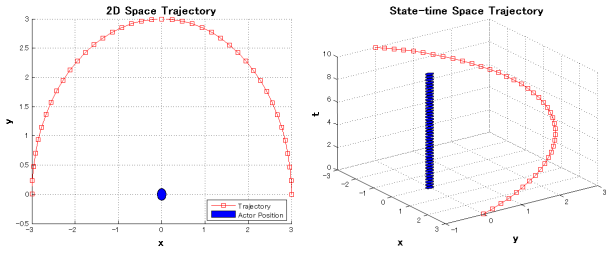
Equation 5, which is the standard MPC, does not consider the final time constraint, *i.e.* the robot $\mathcal{R}$ can possibly reach $\boldsymbol{q_r}_f$ at any time. Another problem which makes the standard MPC not directly applicable to our problem is the state references $\tilde{\boldsymbol{q}}_r$ matter. In the standard MPC, $\tilde{\boldsymbol{q}}_r$ is given as a priori known function and will not be changed once the system runs. Contrary, our cameraman needs to maintain a dynamic reference towards the target, *i.e.* even if the target is moving, our robot has to keep the relative shape of trajectory, which means the reference will be continuously updated based on the target state.

3) Robot Kinematic Model: Here we use an omnidirectional robot model on which a camera is statically attached, which means the camera view will be aligned with the robot orientation. The robot state $\boldsymbol{q} \in \mathbb{Q}$ is described by its center position (q_x, q_y) and orientation (q_θ), forming a tuple $\boldsymbol{q} = \{q_x, q_y, q_\theta\}$. Let the robot control $\boldsymbol{u} \in \mathbb{U}$ be described by a tuple $\boldsymbol{u} = \{u_x, u_y, u_\theta\}$ denoting the translational (u_x, u_y) and angular (u_θ) velocity of the robot, the kinematic model of the robot is then derived as

$$\begin{aligned} \dot{\boldsymbol{q}} &= f(\boldsymbol{q}, \boldsymbol{u}), \\ \begin{bmatrix} \dot{q_x} \\ \dot{q_y} \\ \dot{q_\theta} \end{bmatrix} &= \begin{bmatrix} \cos q_\theta & -\sin q_\theta & 0 \\ \sin q_\theta & \cos q_\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_x \\ u_y \\ u_\theta \end{bmatrix}. \end{aligned} \tag{7}$$

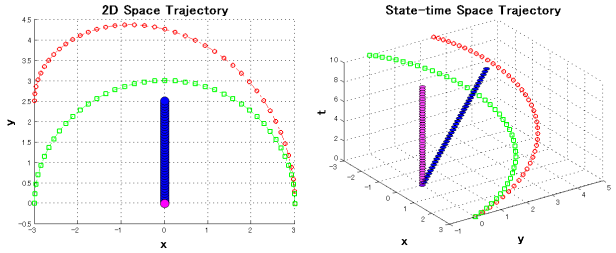
4) Problem Construction: Now we will construct our cameraman robot problem which is an extension of the trajectory tracking problem, by considering the target movement and the final time constraint. We keep the idea of the MPC for taking the first control of the generated optimal control sequences. We then construct our problem as follows:

(a) The final time (T_f) constraint is added by modifying the upper bound of the summation in eq. 5, $(t_0 + T_H)$ into



(a) (b)

Fig. 2: Trajectory: (a) 2D space and (b) state-time space. The red color is the trajectory reference, and the blue one represents the target.



(a) (b)

Fig. 3: Trajectory comparison between static and moving target: (a) 2D space and (b) state-time space.Magenta and blue color represent the static and moving target, while green and red line denote the static and dynamic trajectory references.

$(T_f - t_0)$, which means the horizon of the optimization problem changes over the time.

(b) One of the critical tasks of our cameraman robot is to maintain the relative shape of the trajectory. It means the trajectory will change and follow the target movement. Here we define $f_{q_\mathcal{A}} : \tilde{\boldsymbol{q}}_r \mapsto \tilde{\boldsymbol{q}}_r^{\mathcal{A}}$ as a function which maps the relative trajectory reference $\tilde{\boldsymbol{q}}_r$ to a new trajectory reference $\tilde{\boldsymbol{q}}_r^{\mathcal{A}}$ which is aware of the target state $\boldsymbol{q}_\mathcal{A}$.
The basic idea is to bring the trajectory curve (*i.e.* $\tilde{\boldsymbol{q}}_r$) into a *state-time space* $\mathbb{Q} \times \mathbb{T}$, and then sample it with a specific time step (Δt) using a sampling function $f_s : \tilde{\boldsymbol{q}}_r \mapsto \tilde{\boldsymbol{q}}_{r,s}$, giving

$$\tilde{\boldsymbol{q}}_{r,s} = \{\forall \boldsymbol{q_r}(k\Delta t); k = \{0, 1, \ldots, \frac{T_f}{\Delta t}\}\}, \tag{8}$$

where $\boldsymbol{q_r}(0)$ and $\boldsymbol{q_r}(T_f)$ represent the initial and final state of the trajectory (see Fig. 2).

Definition 1: *Every sampled state of the trajectory ($\boldsymbol{q_r}(k\Delta t)$) retains relative metric properties (e.g. distance, orientation, and time) w.r.t. the target $\mathcal{A}$.*

Using this definition, the dynamic trajectory is considered by first assuming that the target is motionless, from which the relative metric property of each $\boldsymbol{q_r}(k\Delta t)$ can be extracted. We then calculate the target movement and estimate the target poses at each time step. Let $\tilde{\boldsymbol{q}}_\mathcal{A}$ be sequence of the target pose in the space-time when it is motionless, and $\tilde{\boldsymbol{q}}'_\mathcal{A}$ be the estimated one when moves, there exists a set of geometric transformation $G_\mathcal{A}$, so that

$$\tilde{\boldsymbol{q}}'_\mathcal{A} = G_\mathcal{A} \circ \tilde{\boldsymbol{q}}_\mathcal{A}, \tag{9}$$

which is an element-wise product. Prediction of $\tilde{\boldsymbol{q}}'_\mathcal{A}$ will be later explained in the next subsection.
Using the obtained $G_\mathcal{A}$, we then describe the target-aware trajectory (see Fig. 3) $\tilde{\boldsymbol{q}}_{\boldsymbol{r,s}}^{\mathcal{A}} = \{\boldsymbol{q}_{\boldsymbol{r,0}}^{\mathcal{A}}, \boldsymbol{q}_{\boldsymbol{r,1}}^{\mathcal{A}}, \ldots, \boldsymbol{q}_{\boldsymbol{r,f}}^{\mathcal{A}}\}$, given by

$$\begin{aligned} \tilde{\boldsymbol{q}}_{\boldsymbol{r,s}}^{\mathcal{A}} &= f_{q_\mathcal{A}}(\tilde{\boldsymbol{q}}_r) \\ &= G_\mathcal{A} \circ f_s(\tilde{\boldsymbol{q}}_r) \\ &= G_\mathcal{A} \circ \tilde{\boldsymbol{q}}_{r,s}. \end{aligned} \tag{10}$$

(c) Additionally, another constraint is introduced for the cameraman robot problem, which is called *target visibility constraint*. This constraint is a natural consequence of the cameraman robot where the target needs to be seen from the robot camera (*i.e.* not covered by obstacles) as long as possible. Since the visibility is a mutual property, we represent the constraint in the robot state space. Let $\mathcal{V} \subset \mathbb{Q}$ be a set of robot states visible from the target $\mathcal{A}$, we then define the visibility constraint as

$$f_{vis}(\boldsymbol{q}) = \begin{cases} 0, & \forall \boldsymbol{q} \in \mathcal{V} \\ e, & \forall \boldsymbol{q} \notin \mathcal{V} \end{cases} \tag{11}$$

where e is a constant.

(d) Lastly, the camera is needed to always face the target. Since the camera is statically placed on the robot, it means the camera direction will be the same as the robot orientation. We then create a new constraint,

$$\left| q_\theta - \tan^{-1}\left(\frac{q_y - q_{\mathcal{A}y}}{q_x - q_{\mathcal{A}x}}\right) \right| \leq \xi, \tag{12}$$

where $\{q_x, q_y, q_\theta\} \in \boldsymbol{q}$ is the robot state, $\{q_{\mathcal{A}x}, q_{\mathcal{A}y}\} \in \boldsymbol{q}_\mathcal{A}$ is the target state, and ξ is a small constant.

Putting them together, we then define our cameraman robot problem as a dynamic trajectory tracking with the final time constraint, which minimizes the cost function given by

$$\begin{aligned} &\tilde{\boldsymbol{u}}_{\boldsymbol{k}}^* = \min_{\tilde{\boldsymbol{u}}_{\boldsymbol{k}}} J_k \simeq \sum_{k=t_0}^{T_f - t_0} (\ell(\boldsymbol{q_{e_k}'}, \boldsymbol{u}_k^t) + f_{vis}(\boldsymbol{q}_k^t)), \\ &\text{subject to} \\ &\dot{\boldsymbol{q}}^t = f(\boldsymbol{q}^t, \boldsymbol{u}^t);\ \boldsymbol{q}^t \in \mathbb{Q} \times \mathbb{T}, \boldsymbol{u}^t \in \mathbb{U} \times \mathbb{T}, \\ &\boldsymbol{q_{e_k}'} = \boldsymbol{q}_k^t - \boldsymbol{q}_{\boldsymbol{r,k}}^{\boldsymbol{\mathcal{A}}};\ \boldsymbol{q}_{\boldsymbol{r,k}}^{\boldsymbol{\mathcal{A}}} \in \tilde{\boldsymbol{q}}_{\boldsymbol{r,s}}^{\mathcal{A}}, \\ &\left| q_{\theta k}^t - \tan^{-1}\left(\frac{q_{y k}^t - q_{\mathcal{A}y}}{q_{x k}^t - q_{\mathcal{A}x}}\right) \right| \leq \xi, \\ &\boldsymbol{q_{min}} \leq \boldsymbol{q}^t \leq \boldsymbol{q_{max}}, \\ &\boldsymbol{u_{min}} \leq \boldsymbol{u}^t \leq \boldsymbol{u_{max}}. \end{aligned} \tag{13}$$

where $\boldsymbol{q}^t \in \mathbb{Q} \times \mathbb{T}$ and $\boldsymbol{u}^t \in \mathbb{U} \times \mathbb{T}$ show that we are now working in the state-time space.

B. Predicting Trajectory Based on The Target Movement

As already described in the previous subsection, the trajectory given to the robot depends on the actor movement. It is obvious that predicting and estimating a person movement is a difficult task. Here we make an assumption by simplifying the target movement as a constant velocity model.

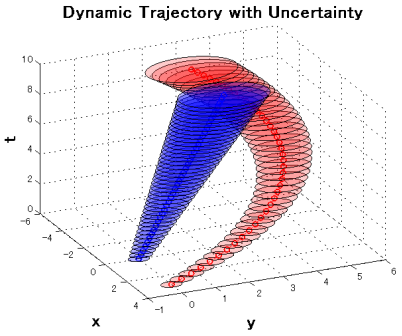


Fig. 4: Dynamic trajectory with uncertainty. The blue cone represents the moving target, while the red one denotes the dynamic trajectory reference.

Let $\mathcal{X} = \{x, y, \theta, v\}$, representing the pose $\{x, y, \theta\}$ and velocity v, be the actor/target $\mathcal{A}$ state, then the target model is defined as follows

$$\mathcal{M}(\mathcal{X}_k) = \begin{cases} x_k &= x_{k-1} + v_{k-1}\cos\theta_{k-1}\delta t + \epsilon_x, \\ y_k &= y_{k-1} + v_{k-1}\sin\theta_{k-1}\delta t + \epsilon_y, \\ \theta_k &= \theta_{k-1} + \epsilon_\theta, \\ v_k &= v_{k-1} + \epsilon_v, \end{cases} \tag{14}$$

where $\{\epsilon_x, \epsilon_y, \epsilon_v, \epsilon_\theta\}$ are gaussian noises. The target movement is then predicted using unscented kalman filter (UKF) [13]. Figure 4 shows the dynamic trajectory with uncertainty and the target movement.

By omitting the detail of the UKF, we get

$$\begin{aligned} \tilde{\mathcal{X}} &= \{\mathcal{X}_k\}(k = 0, \ldots, T_f), \\ \tilde{\Sigma} &= \{\Sigma_k\}(k = 0, \ldots, T_f), \end{aligned} \tag{15}$$

where $\tilde{\mathcal{X}}$ denote the predicted state of the target from the current time to the final time T_f, and $\tilde{\Sigma}$ are its respective covariances. The state of each element in $\tilde{\mathcal{X}}$ is then truncated (by omitting v_k) so that it will have same dimension with $\tilde{\boldsymbol{q}}'_{\mathcal{A}}$ (please see eq. 9). This predicted state of the target is then used for obtaining geometric transformation $G_{\mathcal{A}}$, and maps the relative trajectory reference to the predicted trajectory which consider the target movement (see eq. 10).

III. Robot Control Along Trajectory via State-time Space Stochastic Approach

After explaining the formalities of the cameraman robot problem as pointed out by eq. 13, we will explain the solution of those optimization problems. We propose a stochastic approach for solving our cameraman robot problem, employing a randomized-tree guided by a state-time potential. The guided-stochastic search is chosen in order to manage the calculation time using its incremental behavior [14], as it becomes one of the critical requirement for applying the algorithm to the real system.

Here a modified random tree search algorithm is used, from which we expect a suboptimal result in a reasonable time. The idea is to bring all cost function and constraints into the state-time space, and then samples the possible control sequences. Some implementation considerations of our algorithm will be presented in subsection IV-C. We will also provide comparison with an optimal control solver (here we use pseudospectral method [16]) later.

The algorithm is started by remodeling eq. 13 as follows:

(a) The obstacle constraint $\boldsymbol{q_{min}} \leq \boldsymbol{q}^t \leq \boldsymbol{q_{max}}$ is modified to the *obstacle map*;
(b) The visibility cost $f_{vis}(\boldsymbol{q}^t_k)$ is changed to the *visibility map*;
(c) The trajectory error function $\ell(\boldsymbol{q'_{e_k}}, \boldsymbol{u}^t_k)$ is changed to the *trajectory cost map*;

Each map will be described in the following subsection.

A. Building Cost Maps

Let $\mathbb{C} \simeq \mathbb{Q} \times \mathbb{T}$ representing the state-time space, we build the cost maps as follows:

1) Obstacle Map: Let $\mathcal{F} \subset \mathbb{C}$ be a set of area in the state-time space which is not occupied by any object, the obstacle map is then given by

$$\mathcal{D}(\boldsymbol{q}^t) = \begin{cases} \gamma\|\boldsymbol{q}^t - \boldsymbol{q}^t_{obs}\|, & \text{for } \boldsymbol{q}^t \in \mathcal{F}, \\ 0, & \text{otherwise}, \end{cases} \tag{16}$$

where $\boldsymbol{q}^t_{obs}$ is the nearest obstacle to the state $\boldsymbol{q}^t$, and γ is a constant (currently, $\gamma = 10$).

One of the advantages of using the state-time space is that we can directly consider the moving obstacles inside the obstacle map, by estimating the moving object poses for each time step.

2) Visibility Map: By reusing the notation in eq. 11, let $\mathcal{V} \subset \mathbb{C}$ now denote a set of area visible from the target $\mathcal{A}$ in the certain time step (*e.g.* for $t = t_0$, $\boldsymbol{q}^t \in \mathcal{V}$ is visible from the target $\boldsymbol{q}^t_{\mathcal{A}}$), the visibility map is then defined as

$$\mathcal{Y}(\boldsymbol{q}^t) = \begin{cases} 0, & \text{for } \boldsymbol{q}^t \in \mathcal{V}, \\ e, & \text{otherwise}, \end{cases} \tag{17}$$

where e is a constant (currently, $e = 100$).

3) Trajectory Cost Map: We convert the trajectory error function ($\boldsymbol{q'_{e_k}} = \boldsymbol{q}^t_k - \boldsymbol{q}^{\mathcal{A}}_{\boldsymbol{r,k}}$) into the trajectory cost map $\mathcal{Z} \subset \mathbb{C}$ using generalized logistic function,

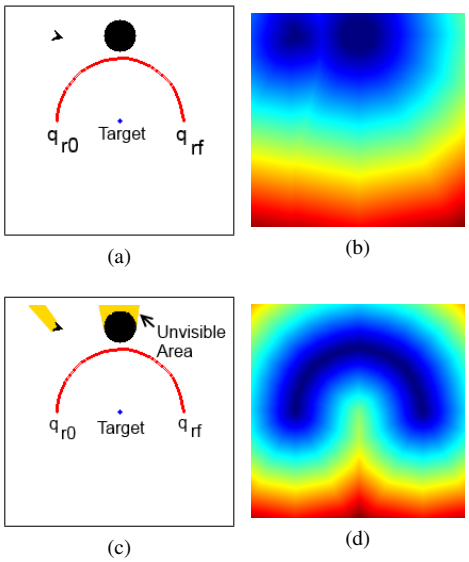
$$\mathcal{Z}(\boldsymbol{q}^t) = \begin{cases} 0, & \text{for } \boldsymbol{q}^t \in \tilde{\boldsymbol{q}}^{\mathcal{A}}_{r,s}, \\ (1 + h\exp^{-\alpha\vartheta(\boldsymbol{q}^t - \tilde{\boldsymbol{q}}^{\mathcal{A}}_{r,s})})^{-\frac{1}{\vartheta}}, & \text{otherwise}, \end{cases} \tag{18}$$

where $(\boldsymbol{q}^t - \tilde{\boldsymbol{q}}^{\mathcal{A}}_{r,s})$ is the distance of the state $\boldsymbol{q}^t$ to the trajectory reference $\tilde{\boldsymbol{q}}^{\mathcal{A}}_{r,s}$, and $\{h, \alpha, \vartheta\}$ are constants (currently, $h = 1, \alpha = 0.1$, and $\vartheta = 0.5$). Equation 18 means that we will keep the robot as close as possible to the reference.

Figure 5 shows each cost map example for one time slice. The black circle and arrow are obstacles, the red line is the trajectory reference, the blue dot is the target position, and $\boldsymbol{q}_{r0}$ and $\boldsymbol{q}_{rf}$ respectively denote the start and end of trajectory reference. The color gradation blue-to-red represents the lower-to-higher value in the map.

B. 3D Time-space Wavefront Propagation-based Stochastic Control Search

1) 3D Time-space Wavefront Propagation Potential: Our idea is to search for the control sequences which optimize



(a) (b) (c) (d)

Fig. 5: Cost maps: (a) original map, (b) obstacle map, (c) visibility map, and (d) trajectory error map.

the cost of the obstacle, visibility and, trajectory maps. Using the cost maps directly for control search has a drawback, *i.e.* the control search may suffer from *local minima*. To avoid those problem, we use a monotonic wavefront in the state-time space, as suggested by [10].

Basically, our monotonic wavefront $\phi(\boldsymbol{q}^t)$ spreads from a source point $\boldsymbol{q}^t(0)$ to all spaces in the configuration by following Eikonal expression

$$\begin{aligned} \phi(\boldsymbol{q}^t) \simeq |\nabla\tau(\boldsymbol{q}^t)| = \frac{1}{\rho(\boldsymbol{q}^t)}, \\ \boldsymbol{q}^t(0) = 0, \end{aligned} \tag{19}$$

where $\tau(\boldsymbol{q}^t)$ denotes the time for the wavefront reaching $\boldsymbol{q}^t$, and $\rho(\boldsymbol{q}^t)$ represents the viscosity of the $\boldsymbol{q}^t$. Here we use $\boldsymbol{q}^t(0) = \boldsymbol{q}^{\mathcal{A}}_{\boldsymbol{r},\boldsymbol{f}}$ (the end of the trajectory) as the source point.

Equation 19 needs the viscosity function $\rho(\boldsymbol{q}^t)$, which determines the speed of the wavefront. For our case, the viscosity function is expected to make the movement of the wavefront faster at the area which is close to the trajectory reference, far from the obstacles, and visible from the target. Here we utilize the previously computed cost maps to build a viscosity function given by following element-wise product

$$\rho(\boldsymbol{q}^t) = \mathcal{D}(\boldsymbol{q}^t) \circ \mathcal{Y}'(\boldsymbol{q}^t) \circ \mathcal{Z}'(\boldsymbol{q}^t), \tag{20}$$

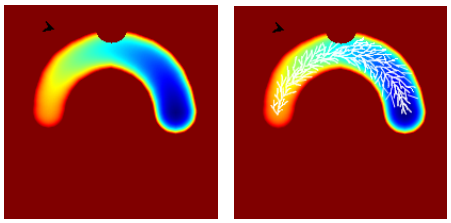
where $\mathcal{Y}'(\boldsymbol{q}^t)$ and $\mathcal{Z}'(\boldsymbol{q}^t)$ are respectively the normalized $\mathcal{Y}(\boldsymbol{q}^t)$ and $\mathcal{Z}(\boldsymbol{q}^t)$ which gives the biggest value to the visible area $\boldsymbol{q}^t \in \mathcal{V}$ and the reference $\mathcal{Z}(\tilde{\boldsymbol{q}}^{\mathcal{A}}_{r,t})$.

Equation 19 can be approximated by the first order finite difference scheme

$$\left(\frac{\tau(\boldsymbol{q}^t) - \tau_1}{\Delta \boldsymbol{q}^t_x}\right)^2 + \left(\frac{\tau(\boldsymbol{q}^t) - \tau_2}{\Delta \boldsymbol{q}^t_y}\right)^2 = \frac{1}{\rho\,(\boldsymbol{q}^t)^2} \tag{21}$$

where

$$\begin{aligned} \tau_1 = \min\left(\tau(\boldsymbol{q}^t_{x+1,y}), \tau(\boldsymbol{q}^t_{x-1,y})\right) \\ \tau_2 = \min\left(\tau(\boldsymbol{q}^t_{x,y+1}), \tau(\boldsymbol{q}^t_{x,y-1})\right) \end{aligned} \tag{22}$$



(a) (b)

Fig. 6: Potential wavefront propagation of the given map in Fig. 5, projected for a time slice: (a) wavefront, and (b) the tree expansion.

The solution of (21) is numerically given by

$$\tau(\boldsymbol{q}^t) = \begin{cases} \tau_1 + \frac{1}{\rho(\boldsymbol{q}^t)} & \text{for } \tau_2 \geq \tau \geq \tau_1 \\ \tau_2 + \frac{1}{\rho(\boldsymbol{q}^t)} & \text{for } \tau_1 \geq \tau \geq \tau_2 \\ \mathcal{Q}(\tau_1, \tau_2, \rho(\boldsymbol{q}^t)) & \text{for } \tau \geq \max(\tau_1, \tau_2) \end{cases} \tag{23}$$

where $\mathcal{Q}(\tau_1, \tau_2, \rho(\boldsymbol{q}^t))$ denote the quadratic solution of eq. 21. Equation 21 and 22 can be solved using a variant of *Fast Iterative Method* (FIM) [10]. Due to the space limitation, the detail of the FIM is omitted, and we encourage the reader to refer to [10] for further explanation.

2) Stochastic Control Search: By using wavefront $\phi(\boldsymbol{q}^t)$ (see Fig. 6a) as a potential, in fact we can easily extract the optimal control sequences using *steepest descent* search.

The problem is that the dynamic trajectory $\tilde{\boldsymbol{q}}^{\mathcal{A}}_{\boldsymbol{r},\boldsymbol{t}}$ holds uncertainties due to the target movement prediction, as suggested by eq. 15. To deal with this problem, we then propose a stochastic control search utilizing the wavefront potential $\phi(\boldsymbol{q}^t)$ as the guidance. We use the 3D time-space randomized tree algorithm introduced in [10] with a modification.

In short, the algorithm can be summarized as follows:

(a) Initialize the threshold s_{th} by the value of $\phi(\boldsymbol{q}^t_0)$ and $t_{now} = t_0$;
(b) Pick a random point $\boldsymbol{q}^t_{rand}$ in $t > t_{now}$ so that $\phi(\boldsymbol{q}^t_{rand})$ has better value than s_{th};
(c) Find the nearest node $\boldsymbol{q}^t_{near}$ to the $\boldsymbol{q}^t_{rand}$, extend it using the control law, and update $t_{now} = t_{now} + \Delta t$, or $t_{now} = t_0$ for $t_{now} = T_f$;
(d) Update s_{th} using following policies,

$$s_{th} = \begin{cases} s_{th} - \frac{1}{2}(\phi(\boldsymbol{q}^t_{rand}) - s_{th}), & \text{for } t_{now} < T_f \\ \phi(\boldsymbol{q}^t_0), & \text{otherwise;} \end{cases} \tag{24}$$

(e) Repeat to (b) until the allocated time is reached.

The policies (c) and (d) above make sure the convergence of the tree (see Fig. 6b) without losing the dispersion while maintaining the final time constraint.

One notable thing in the algorithm above is the policy (c). The original 3D time-space randomized tree search for the best control from a set of predefined motion controls for extending the tree. In our cameraman robot case, applying the same method has a drawback. The weakness is due to the control dimension (the cameraman robot needs three control components $\boldsymbol{u} = \{u_x, u_y, u_\theta\}$ and one controlled parameter, *i.e.* the robot orientation should face to the target) which will slow down the searching process in the tree expansion.

To escape from the weakness, rather than searching from a set of motion controls, we use directly control laws for extending the tree. The use of omnidirectional robot gives us possibility to determine the control law for the robot translation and orientation separately [15]. Here we adopt [15] to use *exponential gain* control for the robot translation and the *Proportional-Derivative* (PD) control for the robot orientation, as follows,

$$\begin{aligned} u_x &= v_d \cos(\arctan(-K_r r) + \theta_p), \\ u_y &= v_d \sin(\arctan(-K_r r) + \theta_p), \\ u_\theta &= K_p e_\theta + K_d \frac{d}{dt} e_\theta, \end{aligned} \tag{25}$$

where r is the distance of the robot pose to the reference, v_d denote the desired velocity, e_θ is the orientation error towards the target, θ_p is tangent direction of the robot to the reference, and $\{K_r, K_p, K_d\}$ respectively denote the constant of translation, proportional, and derivative gains (currently, $K_r = 2, K_p = 4$, and $K_d = 3.5$).

C. Speeding Up the Algorithm

It is obvious that calculating $\phi(\boldsymbol{q}^t)$ takes a significant amount of time. We then come with an idea, from the fact that the control search is basically done at a limited area in the $\phi(\boldsymbol{q}^t)$, which is around the trajectory reference. To reduce the computational time, we apply a boundary area from which the wavefront potential, as well as the control search area, is limited. We take advantage the uncertainty ellipses area of the trajectory which depends on the target motion (please see eq. 15).

Let $\mathcal{H} \subset \mathbb{C}$ denote the restricted area formed by the uncertainty ellipses $\tilde{\Sigma}$ in the state-time space, so that

$$\mathcal{H} = \bigcup_{k=0}^{T_f} \{\forall \boldsymbol{q}^t | \boldsymbol{q}^t \in \Sigma_k\}. \tag{26}$$

We then build an additional cost map representing the area limitation as follows,

$$\mathcal{S}(\boldsymbol{q}^t) = \begin{cases} 1, & \text{for } \boldsymbol{q}^t \in \mathcal{H}, \\ 0, & \text{otherwise.} \end{cases} \tag{27}$$

The viscosity function in eq. 20 is then modified to

$$\rho(\boldsymbol{q}^t) = \mathcal{D}(\boldsymbol{q}^t) \circ \mathcal{Y}'(\boldsymbol{q}^t) \circ \mathcal{Z}'(\boldsymbol{q}^t) \circ \mathcal{S}(\boldsymbol{q}^t), \tag{28}$$

so that we can calculate $\phi(\boldsymbol{q}^t)$ for a narrower area.

IV. Experiments and Discussions

We verify our algorithm using a 3D simulator and compare the performance with the Optimal Control Problem (OCP) solver. The algorithm is implemented in C++ and runs on a Windows laptop (Core i7, 2.4 GHz). The map used for experiments is discretized by 10 cm resolutions and the map size depends on the size of given trajectory.

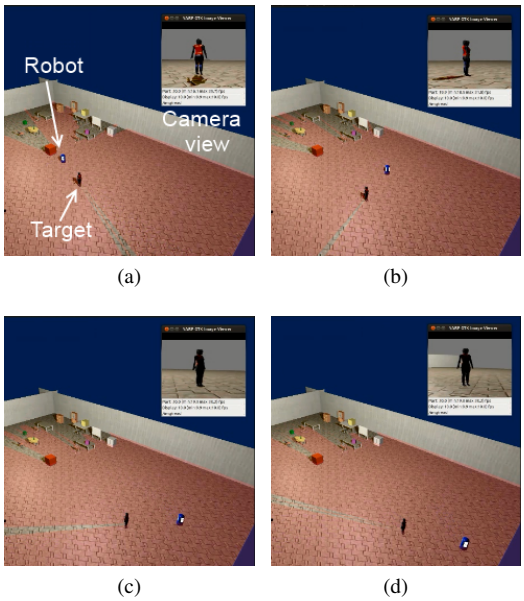


Fig. 7: Snapshots of the cameraman robot simulation.

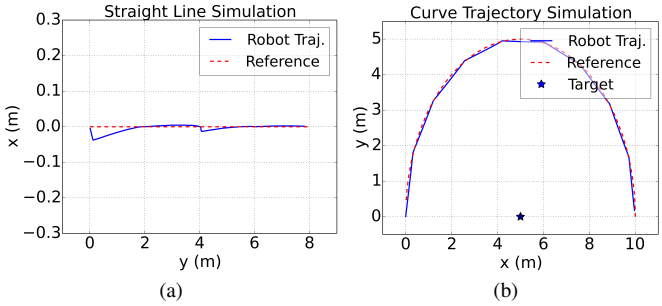


Fig. 8: Executed robot trajectory for the static target: (a) straight line and (b) curve line. The red dashes denote the given reference, the black dot is the target position, and the blue line is the robot trajectory.

A. Simulation

We use MORSE 3D Simulator [11] which runs on a Linux PC (Core i5, 2.1 GHz) and interconnected with the cameraman robot algorithm using socket communication [12]. We test our algorithm using straight and curve trajectory references with a static and moving target. A feedback controller running at 10 Hz is used for executing the control commands from our algorithm. We give the final time constraint of 20 seconds for the straight line and 24 seconds for the curve trajectory. Figure 7 shows the example of running simulation[2].

We first evaluate the algorithm using a static target for the straight and curve trajectories without obstacles. In the next simulation, the target moves along *x-axis* with the speed of 0.4 m/s. Lastly, we assess the influence of adding an obstacle to the algorithm performance.

Figure 8 shows the executed trajectories for the static target cases. We can see that the robot tracks the given trajectories well. The robot also maintains the orientation towards the target and fulfills the final time constraint with

[2]An accompanying video showing the full simulation is provided

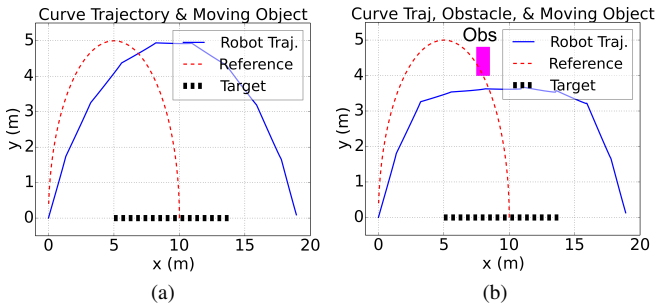


Fig. 9: Executed robot trajectory for the moving target: (a) without obstacle and (b) with obstacle. The red dashes denote the given relative reference, the black dots are the target movement, the blue line is the robot trajectory.

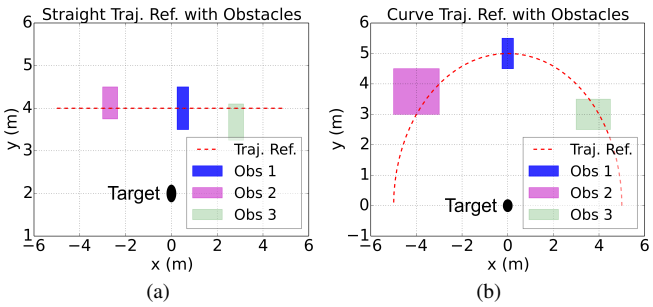


Fig. 10: The given trajectory references and obstacles for comparison: (a) straight line and (b) curve line.

TABLE I: Final Time and Orientation Error

	Final Time Error (s)	Orientation Error (deg.)	
		avg.	std. dev.
straight + static target	0.25	2.45	0.78
curve + static target	0.3	2.98	1.31
curve + moving target	0.55	4.36	2.56
curve + moving target + obstacle	0.6	4.64	2.77

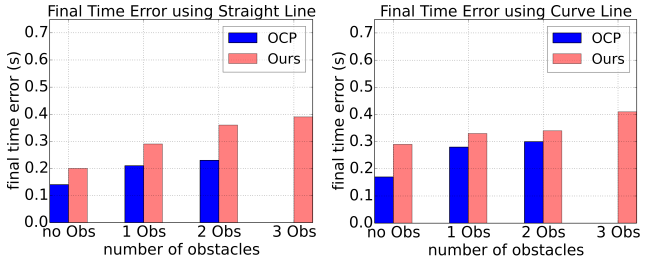


Fig. 12: Comparison of final time error: (a) straight line and (b) curve line. The blue bar denotes the OCP results, while the red is our algorithm.

the smallest error, as shown in the table I.

When the target moves, our algorithm successfully maintains the relative trajectory as shown in Fig. 9a. By adding an obstacle, the robot trajectory obviously deviates from the reference, nevertheless the algorithm manages to keep the visibility by choosing the inner curve as shown in the Fig. 9b.

From table I, the cases with moving target tend to give larger error for both final time and orientation error. It is reasonable because the prediction of the moving target holds uncertainties. But overall, the errors are in the reasonable range for saying that the algorithm can solve the problem.

Table II shows the calculation time needed for each step in our algorithm. With the total time of 0.845 seconds for each iteration, our algorithm is fast enough to run real-time on a real robot. One thing should be noted here that the calculation time of the stochastic control search above produces around 750-1000 nodes. We will show in next subsection that our algorithm has incremental behavior, so that the result will be better if we add more time. Obviously, we limit the control search to 0.5 seconds with a trade-off between speed and quality.

TABLE II: Calculation Time Breakdown of Our Algorithm

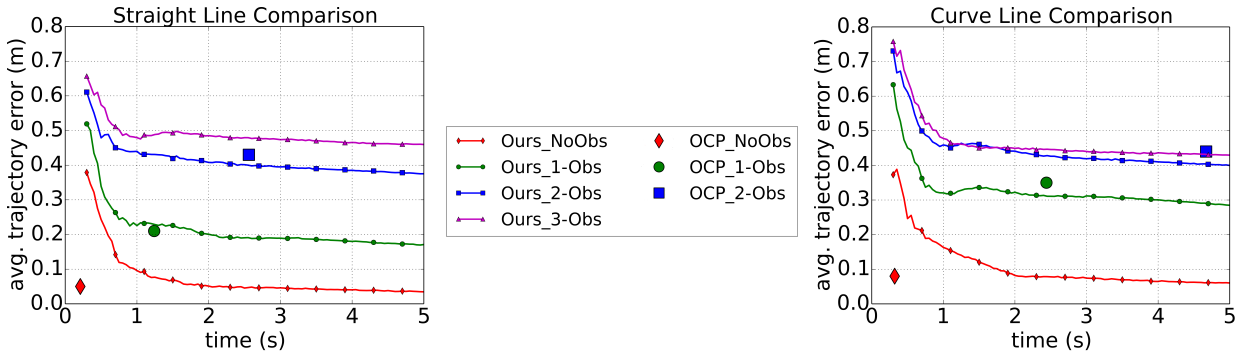
	Calculation Time (s)
Data acquisition and prediction	0.01
Building cost maps	0.035
Wavefront propagation	0.3
Stochastic control search	0.5
Total	0.845

B. Comparison with The OCP Solver

We compare the OCP solver [16] and our algorithm performance for solving the eq. 13. The algorithms are tested using the straight and curve trajectory references with the number of obstacles ranged from zero to three and motionless target, as shown in Fig. 10, with the desired final time is 20 seconds. We do not use the moving target for comparison, because the solver [16] does not handle uncertainties (it will be further explained in IV-C.3). Please note that for the OCP solver, the visibility cost is converted into the obstacle constraints (*i.e.* the area which is not visible from the target is considered as obstacle) for easier representation for the solver.

Figure 11 shows the averaged trajectory error of both methods created at the robot initial position. Our algorithm has incremental behavior, *i.e.* the result is getting better by the time, due to the use of the guided-randomized search. Without any obstacle constraint, the numerical-based OCP can quickly solve the problem with a small trajectory error. Our algorithm has advantages when the number of obstacle is increased, where it quickly converges and achieves similar trajectory error with the OCP solver in faster time. Moreover, the OCP solver even fails to solve the equation in the case of three obstacles are added (it is also the reason why there is no result for the OCP with three obstacles in the Fig. 11).

We then compare the final time error of both methods in the Fig. 12. The final time error is obtained by calculating the difference of the time when the robot reaches the end of trajectory and the desired time. Although the OCP solver always has a better error, the 0.15 seconds difference (at max) of the final time error is considerably small compared



(a) (b) (c)

Fig. 11: Comparison of average trajectory error vs calculation time: (a) straight line, (b) symbol's information, and (c) curve line. Lines denote our algorithm and single markers are the OCP results. The red, green, blue, and magenta colors respectively denote 0, 1, 2, and 3 obstacles exist in the map.

to the desired final time 20 seconds, moreover if we also take account the calculation time as shown in the Fig. 11.

C. Some Remarks

We present some considerations about our cameraman robot system, as follows:

1) Control Constraint vs Final Time: There is possibility when the given final time is too small, then the robot cannot catch up with it no matter if the control is applied. To avoid such constraint violation, here we assume that the given final time constraint is long enough compared to the robot control constraint. We will examine those problems for the future work.

2) Camera Visibility Against Lower Objects: Currently we treat all of the obstacles to have covering behavior, *i.e.* the cameraman robot cannot see the target from behind of the obstacles. In the real cases, there exist some small obstacles which block the path of the robot, but actually do not cover the camera view. These problems will be investigated later.

3) Comparison using Moving Target: As indicated by the previous subsection, we compare our algorithm for the motionless target cases only, because our intention is to clarify the performance of our algorithm against the numerical solver (*i.e.* how fast our algorithm runs while maintaining the trajectory quality close to the one calculated numerically). As the solver [16] does not handle uncertainties, the comparison using a moving target becomes infeasible. Other possible solutions of the moving target cases of eq. 13 are opened for the interested readers.

V. Conclusion

We have presented a novel solution for a cameraman robot problem described as the dynamic trajectory tracking with a final time constraint. By considering the target movement, visibility, and obstacles, we have successfully built a robotic system which imitates one of the jobs of the cameraman using the state-time space stochastic approach.

Regardless of the good performances shown in the experiment section, we still use the static camera. In the real situation, the cameraman person often plays with the zoom and focus of the camera. The future direction of this research lies on the integration of the camera zoom and focus into the system. We will also investigate the problem arises in the sub section IV-C.

References

[1] G. Millerson and J. Owens. "Video Production Handbook". Focal Press.

[2] I. D. Cowling, J. F. Whidborne, and A. K. Cooke. "Optimal trajectory planning and LQR control for a quadrotor UAV". In Int. Control Conference, 2006.

[3] S. G. Vougioukas. "Reactive Trajectory Tracking for Mobile Robots based on Non Linear Model Predictive Control". In IEEE Int. Conf. on Robotics and Automation, pp. 3074-3079, 2007.

[4] T. Faulwasser, B. Kern, and R. Findeisen. "Model Predictive Path-Following for Constrained Nonlinear Systems". In IEEE Int. Conf. on Decision and Control, pp. 8642-8647, 2009.

[5] K. Kanjanawanishkul, M. Hofmeister, and A. Zell. "Path following with an optimal forward velocity for a mobile robot". In 7th IFAC Symp. on Intell. Autonomous Vehicles, pp. 19-24. 2010.

[6] A. Alessandretti, A. P. Aguiar, and C. N. Jones. "Trajectory-tracking and path-following controllers for constrained underactuated vehicles using Model Predictive Control". In European Control Conference, pp. 1371-1376, 2013.

[7] K. Kanjanawanishkul and A. Zell. "Path Following for an Omni directional Mobile Robot Based on Model Predictive Control". In IEEE Int. Conf. on Robotics and Automation, pp. 3341-3346, 2009.

[8] H. Lim, Y. Kang, C. Kim, J. Kim, and B. You. "Nonlinear Model Predictive Controller Design with Obstacle Avoidance for a Mobile Robot". In IEEE/ASME Int. Conf. on Mechatronic Embedded Systems and Applications, pp. 494-499, 2008.

[9] P. A. Nino-Suarez. "Discrete-time sliding mode path-tracking control for a wheeled mobile robot". In IEEE Int. Conf. on Decision and Control, pp. 3052-3057, 2006.

[10] I. Ardiyanto and J. Miura, "3D Time-space Path Planning Algorithm in Dynamic Environment Utilizing Arrival Time Field and Heuristically Randomized Tree", In IEEE Int. Conf. on Robotics and Automation, pp. 187-192, 2012.

[11] G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan. "Modular Open Robots Simulation Engine: MORSE". In IEEE Int. Conf. on Robotics and Automation, pp. 46-51, 2011.

[12] I. Ardiyanto and J. Miura. "RT Components for using MORSE Realistic Simulator for Robotics". The 13th SICE System Integration Division Annual Conference, pp. 535-538, 2012.

[13] S. J. Julier and J. K. Uhlmann. "Unscented filtering and nonlinear estimation". In Proc. The IEEE, vol. 92, no. 3, pp. 401-422, 2004.

[14] I. Ardiyanto and J. Miura. "Real-time navigation using randomized kinodynamic planning with arrival time field". Robotics and Autonomous Systems, Vol.60(12), pp. 1579-1591, 2012.

[15] X. Li, M. Wang, and A. Zell, "Dribbling control of omnidirectional soccer robots". In Proc. IEEE Int. Conf. on Robotics and Automation, pp. 2623-2628, 2007.

[16] V. M. Becerra. "Solving complex optimal control problems at no cost with PSOPT". In IEEE Multi-conf. on Systems and Control, pp. 1391-1396, 2010.