Journal of Mechanical Science and Technology 21 (2007) 1973~1985

Journal of
Mechanical
Science and
Technology

Fast design of the QP-based optimal trajectory for a motion simulator

Young Man Cho*, Hwa Soo Kim, Ik Kyu Kim, Jong Jin Woo and Jongwon Kim

School of Mechanical and Aerospace Engineering, Seoul National Univ., Seoul, 151-744, Korea

(Manuscript Received October 16, 2006; Revised August 20, 2007; Accepted August 20, 2007)

Abstract

The main difficulty in realizing a motion simulator comes from the constraints on its workspace. The so-called washout filter prevents a simulator from being driven to go off its pre-determined boundaries and generate excessive torques. By noting that the existing washout filters are conservative and more aggressive motions may be accommodated, this paper presents a novel approach that fully exploits the simulator workspace and thereby reproduces the real-world sensations with high fidelity. The washout filter converts the real-world input trajectory as a realizable one that satisfies the spatial and dynamic constraints while minimizing the sensation error and fidelity between the motions experienced in the real world and on the motion simulator. The control objective is to reduce the computational burdens by using the QP algorithm. The proposed approach formulates the task of designing a washout filter as a quadratic programming (QP). The direct approach to the solution of the QP often results in a computational burden that amounts to $O(N^3)$ flops and $O(N^2)$ storage space ($N = 10^4 \sim 10^5$, typically). By judiciously exploiting the Toeplitz structures of the underlying matrices, an orders-of-magnitude faster algorithm is obtained to reduce the computational burdens to $O(N \log_2 N)$ flops and $O(N)$ storage space. The extensive simulation studies on the Eclipse-II motion simulator at Seoul National University assure that the QP-based fast algorithm outperforms the existing ones in reproducing the real-world sensations.

Keywords: Motion simulator; Washout filter; Workspace; Linear quadratic regulator (LQR); Quadratic programming (QP); Toeplitz; FFT; Eclipse-II

1. Introduction

For last thirty years, numerous types of motion simulators have been developed to serve different needs. Although pilots and crews have been their primary beneficiaries, the motion simulators have been steadily expanding their presences in novel applications such as prototype testing, human behavior study, etc. [1]. Recently, the motion simulators have pioneered their territory into the amusement park to replace the bulky and costly rides [2]. However, the current amusement simulators focus on the visual and audio systems rather than precise reproduction of motions [3]. As a result, the amusement industry has yet to observe a full-fledged motion simulator that provides satisfactory motion cues.

The limited workspace of a motion simulator does not allow direct duplication of the real-world motion on the simulator platform, which naturally prompts the development of the washout filter (whose name originates from the fact that one of its functions is to “wash out” the position of the simulator back to its neutral position). The washout filter converts the real-world motion into the realizable motion on the simulator while minimizing the sensational difference between the real-world and simulated motions [4]. Among many ways of designing washout filters, the so-called classical washout algorithm is most widely accepted by virtue of its simplicity and intuitiveness [1]. Although it essentially provides high-pass filtering of linear acceleration and angular rate, the classical washout algorithm also realizes the sustained (low

*Corresponding author. Tel.: +82 2 880 1644, Fax.: +82 2 883 1513
E-mail address: ymcho85@snu.ac.kr

frequency) specific force cue (linear acceleration) by tilting the motion platform, while exploiting the otolith system's inability to distinguish between pitch motion and longitudinal specific force. Despite its reasonable performance, the algorithm still has to rely on manual tuning of fixed gain parameters [5, 6]. Then emerge adaptive algorithms that continuously update the gains in an effort to minimize the motion errors and the magnitudes of the simulator states [7]. Yet, adaptive algorithms generate some false motion cues. In order to overcome the shortcomings of the classical and adaptive algorithms, a rigorous framework is proposed based on the linear quadratic regulator (LQR) theory [8]. While explicitly considering the human vestibular system, the LQR-based approach factors sensation error, the simulation state, and the command trajectory into the cost function and solves the resulting optimization problem using commercially available tools [8].

Despite its remarkable success, the LQR-based washout filter still leaves room for further improvement on two accounts: the design procedure is iterative and the solution is rather conservative. In other words, it does not take full advantage of the simulator workspace and other constraints since the design procedure takes into account the target trajectory only after one-iteration is completed [8]. These two drawbacks naturally call for an algorithm that explicitly takes into account the simulator constraints at the stage of the problem formulation and non-iteratively finds an optimal solution that utilizes the simulator capability to full extent. When the simulator trajectory is given *a priori*, the constrained quadratic programming (QP) provides a mathematical foundation for such an algorithm. The simulator constraints such as workspace boundary, torque limits etc., are factored into linear matrix inequalities while the sensation error becomes a quadratic cost function.

Despite the excellent performance and clear advantages over the existing approaches, the proposed algorithm suffers from a huge computational burden that does not come into the picture as long as the problem size is small. The conventional solution to the optimization problem (QP) requires $O(N^3)$ flops and $O(N^2)$ storage space, where N is the number dependent upon simulation or ride duration. Considering that typical applications require N ranging from 10^4 to 10^5, the proposed algorithm must overcome the apparent *cul-de-sac* in order to be commercially viable. Direct solution of the constrained QP involves constructing matrices for the quadratic cost and computing several matrix-vector products, which are largely responsible for huge computational burden. By judiciously exploiting the structures of the underlying matrices, it is shown that an orders-of-magnitude faster implementation is possible with a much less storage requirement. The Toeplitz structure (when computing the cost matrices and matrix-vector products) allows such time and storage savings.

Extensive simulation studies are conducted to test the viability of the proposed algorithm with the Eclipse-II motion simulator which allows the full-fledged 6 degrees-of-freedom motions, *i.e.,* infinite rotations as well as finite translations [9, 10]. For smooth trajectories that do not require violating the simulator constraints, two approaches based on the LQR and QP produce similar results. However, for those trajectories that push the limits of the simulator constraints, the QP-based approach displays its clear edge over the LQR-based counterpart in terms of sensation errors.

This paper is organized as follows. Section 2 describes the LQR-based washout filter. Section 3 shows how the problem of finding an optimal trajectory can be recast into a constrained QP. Section 3 also derives an orders-of-magnitude faster algorithm for solving the constrained QP. After briefly explaining the Eclipse-II motion simulator, Section 4 presents the results of simulation studies to examine the performance of the proposed algorithm, which proves its viability in the real-world applications.

2. LQR-based washout filter

In this section, the washout filter design methodology based on the LQR theory is briefly summarized [8], which serves as a foundation for the QP-based design of an optimal trajectory in Section 3. Fig. 1 shows the problem structure for an optimal washout filter design, adopted throughout this paper. The main idea of the LQR-based washout filter design is to find a linear transfer matrix $\mathbf{W}(s)$ that minimizes a certain quadratic cost involving the sensation error e and the simulator input u_s without violating the simulator constraints. The resulting filtering equation is $U_S(s) = \mathbf{W}(s)U_a(s)$ where $U_a(s)$ and $U_s(s)$ are the Laplace transforms of the actual and the washed-out (filtered) trajectories, respectively. Although the algorithms are developed and tested along the longi

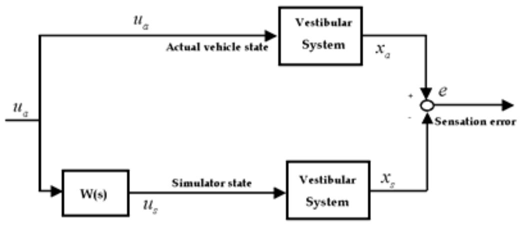

Fig. 1. Problem structure for an optimal washout filter.

tudinal axis only (pitch and surge) throughout this paper, they can be readily generalized to include other axes.

The LQR-based washout filter design begins with a mathematical model of the human vestibular system. Assume that the input u to the vestibular system consists of the angular rate $\dot{\theta}$ and the specific force a_x so that $u = \begin{bmatrix} \dot{\theta} & a_x \end{bmatrix}^T$. Then, the sensed rotational motion $\hat{q}$ (pitch) is given by the mathematical model of the semicircular canals:

$$\begin{aligned}\hat{q} &= \frac{G_s \tau_a s^2 (1+\tau_L s)}{(1+\tau_a s)(1+\tau_1 s)(1+\tau_2 s)}\dot{\theta} \\ &= \frac{T_3 s^3 + T_4 s^2}{s^3 + T_2 s^2 + T_1 s + T_0}\dot{\theta}\end{aligned} \tag{1}$$

where $T_0 = \dfrac{1}{\tau_a \tau_1 \tau_2}$, $T_1 = \dfrac{\tau_a + \tau_1 + \tau_2}{\tau_a \tau_1 \tau_2}$,

$T_2 = \dfrac{\tau_1 \tau_2 + \tau_a(\tau_1 + \tau_2)}{\tau_a \tau_1 \tau_2}$, $T_3 = G_s \tau_a \tau_L T_0$,

$T_4 = G_s \tau_a T_0$ and τ_1 and τ_2 are time constant, with $\tau_1 \gg \tau_2$. τ_a is the adaptation time constant, and τ_L is time constant with the additional lead component. The term G_s defines the static sensitivity in terms of afferent firing rate per unit of acceleration. [8, 11, 12]. Eq. (1) may be written in the state space equation as

$$\begin{aligned}\dot{x}_{1\text{-}3} &= \mathbf{A}_{scc} x_{1\text{-}3} + \mathbf{B}_{scc} u \\ \hat{q} &= \mathbf{C}_{scc} x_{1\text{-}3} + \mathbf{D}_{scc} u,\end{aligned} \tag{2}$$

where $x_{1\text{-}3} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T \in R^3$,

$$\mathbf{A}_{scc} = \begin{bmatrix} -T_2 & 1 & 0 \\ -T_1 & 0 & 1 \\ -T_0 & 0 & 0 \end{bmatrix}, \quad \mathbf{B}_{scc} = \begin{bmatrix} T_4 - T_2 T_3 & 0 \\ -T_1 T_3 & 0 \\ -T_0 T_3 & 0 \end{bmatrix},$$

$\mathbf{C}_{scc} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$ and $\mathbf{D}_{scc} = \begin{bmatrix} T_3 & 0 \end{bmatrix}$.

The sensed specific force $\hat{a}_x$ is related to the input specific force a_x by the otolith model:

$$\hat{a}_x = G_0 \frac{s + A_0}{(s + B_0)(s + B_1)} a_x, \tag{3}$$

where G_0, A_0, B_0, and B_1 are physical constants [8, 12]. The input specific force a_x is assumed to be in the linear combination of three components like

$$a_x = f_x + g\theta - R_{sz}\ddot{\theta} \tag{4}$$

where R_{sz} is the radius from the motion platform centroid to the pilot's head. The first term denotes the acceleration from a linear motion, the second term stems from the gravitational force due to the tilting motion and the third term accounts for the inertia force from the angular acceleration. Eq. (3) may be transformed into the Laplace domain:

$$a_x(s) = f_x(s) + (\frac{g}{s} - R_{sz}s)\dot{\theta} \tag{5}$$

With Eq. (5), the sensed specific force in Eq. (3) is recast into the following form

$$\hat{a}_x = G_0 \begin{bmatrix} \dfrac{-R_{sz}s^3 - R_{sz}A_o s^2 + gs + gA_o}{s(s+B_o)(s+B_1)} & \dfrac{(s+A_o)}{(s+B_o)(s+B_1)} \end{bmatrix} u \tag{6}$$

which is written in the state equation as

$$\begin{aligned}\dot{x}_{4\text{-}8} &= \mathbf{A}_{oto} x_{4\text{-}8} + \mathbf{B}_{oto} u \\ \hat{a}_x &= \mathbf{C}_{oto} x_{4\text{-}8} + \mathbf{D}_{oto} u,\end{aligned} \tag{7}$$

where $x_{4\text{-}8} = \begin{bmatrix} x_4 & x_5 & x_6 & x_7 & x_8 \end{bmatrix}^T \in R^5$,

$$\mathbf{A}_{oto} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ -b & -a & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -b & -a \end{bmatrix},$$

$$\mathbf{B}_{tot} = \begin{bmatrix} c & 0 \\ d-ac & 0 \\ e & 0 \\ 0 & f \\ 0 & h-af \end{bmatrix}, \ \mathbf{C}_{ott} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \end{bmatrix},$$

$\mathbf{D}_{oto} = G_0 \begin{bmatrix} -R_{sz} & 0 \end{bmatrix}$, $a = B_0 + B_1$, $b = B_0 B_1$, $c = G_0 R_{sz}(a - A_0)$, $d = G_0(g + R_{sz}b)$, $e = G_0 g A_0$, $f = G_0$ and $h = G_0 A_0$.

The vestibular model Eq. (2) and Eq. (7) may be combined and transformed into the following state-space description [13]:

$$\dot{x}_v = \mathbf{A}_v x_v + \mathbf{B}_v u$$
$$y_v = \mathbf{C}_v x_v + \mathbf{D}_v u,$$

where $x_v = [x_{1\text{-}3}^T \ x_{4\text{-}8}^T]^T \in \mathrm{R}^8$, $u \in \mathbb{R}^2$, $y_v = \begin{bmatrix} \hat{q} & a_x \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^2$, $\mathbf{A}_v = \begin{bmatrix} \mathbf{A}_{scc} & 0 \\ 0 & \mathbf{A}_{oto} \end{bmatrix}$, $\mathbf{B}_v = \begin{bmatrix} \mathbf{B}_{scc} \\ \mathbf{B}_{oto} \end{bmatrix}$, $\mathbf{C}_v = \begin{bmatrix} \mathbf{C}_{scc} & 0 \\ 0 & \mathbf{C}_{oto} \end{bmatrix}$ and $\mathbf{D}_v = \begin{bmatrix} \mathbf{D}_{scc} \\ \mathbf{D}_{oto} \end{bmatrix}$ [8].

Denoting the vestibular system states from the actual ride and the simulator as x_a and x_s and defining the corresponding state error x_e as $x_e \triangleq x_s - x_a$ yields the following state-space description for the sensation error e:

$$\begin{aligned} \dot{x}_e &= \mathbf{A}_v x_e + \mathbf{B}_v u_s - \mathbf{B}_v u_a \\ e &= \mathbf{C}_v x_e + \mathbf{D}_v u_s - \mathbf{D}_v u_a, \end{aligned} \tag{8}$$

where $u_a = \begin{bmatrix} \dot{\theta}^a & a_x^a \end{bmatrix}^{\mathrm{T}}$ and $u_s = \begin{bmatrix} \dot{\theta}^s & a_x^s \end{bmatrix}^{\mathrm{T}}$.

In addition to the sensation error, the washout filter design must take into account the simulator state x_d, which is needed to guarantee that the simulator does not violate its constraints. Define the simulator state x_d as $\begin{bmatrix} \iiint a_x^s dt^3 & \iint a_x^s dt^2 & \int a_x^s dt & \theta^s \end{bmatrix}^{\mathrm{T}}$, where a_x^s and θ^s are the specific force and angular rate in the simulator, respectively. Then, the dynamics of the simulator state may be given by

$$\dot{x}_d = \mathbf{A}_d x_d + \mathbf{B}_d u_s \tag{9}$$

where $\mathbf{A}_d = \begin{bmatrix} 0&1&0&0&0 \\ 0&0&1&0&0 \\ 0&0&0&0&0 \\ 0&0&0&0&1 \\ 0&0&0&0&0 \end{bmatrix}$ and $\mathbf{B}_d = \begin{bmatrix} 0&0 \\ 0&0 \\ 0&1 \\ 0&0 \\ 1&0 \end{bmatrix}$.

Now, equipped with the dynamic models Eqs. (8) and (9) for sensation error and simulator state, the problem of designing an optimal washout filter is cast into the framework of the LQR theory [8]:

Find $u_s(t)$ over $[t_0, t_1]$ that minimizes the cost J given by

$$J = \int_{t_0}^{t_1} (e^T \mathbf{Q} e + u_s^{\mathrm{T}} \mathbf{R} u_s + x_d^{\mathrm{T}} \mathbf{R}_d x_d) dt \tag{10}$$

where $[t_0, t_1]$ is the simulator ride duration (or simulation time), $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{R}_d$ are the weighting matrices. It is well-known that the solution to the above optimization problem may be found by solving a Riccati equation with a commercial package, *e.g.* MATLAB [13, 14], which results in the transfer matrix $\mathbf{W}(s)$. In practice, the procedure of designing an optimal washout filter based on the LQR theory still requires a few iterations. The weighting matrices $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{R}_d$ are first determined and the corresponding filter $\mathbf{W}(s)$ is found, which generates u_s from u_a. The procedure is iterated until u_s satisfies all the simulator constraints:

$$\left|\dot{\theta}^s(t)\right| \le \dot{\theta}^s_{\max},\ \left|a_x^s(t)\right| \le a^s_{x,\max},\ \left|\theta^s(t)\right| \le \theta^s_{\max},$$
$$\left|v^s(t)\right| \le v^{\mathrm{s}}_{\max}, \text{ and } \left|d^s(t)\right| \le d^s_{\max} \text{ for } t \in [t_0, t_1]$$

where $\dot{\theta}^s_{\max}$, $a^s_{x,\max}$, $\theta^s_{\max}$, $v^{\mathrm{s}}_{\max}$, and $d^s_{\max}$ are maximum allowable simulator angular velocity, specific force (or acceleration), angle, velocity and displacement, respectively.

3. Fast design of the QP-based optimal trajectory

The LQR-based washout filter in Section 2 generates filtered trajectories for the simulator only after the filter is designed with the corresponding weights $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{R}_d$. As a result, it is also checked after the filter design whether the designed (or filtered) trajectory satisfies the simulator constraints or not. Despite iterative weighting-tunning, it is inevitable that the resulting washout filter be conservative, *i.e.*, does not fully utilize the simulator capability. Of course, when a motion simulator allows interactive ride through devices like a joystick, the washout filter must compute the filtered trajectory during the simulator run with a pre-designed filter in order to reduce the computational complexity, which naturally calls for a conservative washout filter. Yet, in many applications where the trajectory is given *a priori*, it is possible to design a washout filter or a filtered trajectory that better utilizes the simulator capability and consequently reduces the sensation error to lower level. In this section, an algorithm based on QP is proposed to generate such a trajectory while explicitly taking into account the simulator constraints at the stage of problem formulation. Although its applicability is slightly limited, the proposed approach delivers better performance than the LQR-based one when the trajectory is given *a priori*.

3.1 Optimal trajectory design via constrained QP

This subsection begins with reformulating the optimization problem Eq. (10) in discrete time. The justification comes from two accounts:

1. When a trajectory is given *a priori*, it is generally described and stored in discrete time.
2. The optimization problem in discrete time renders a rather nice numerical solution via QP.

First, sampling u_s at the sampling period $t_s=(t_1-t_0)/N$ gives $u_s(k)=u_s(t_0+kt_s)$ for $k=0,1,\cdots,N$. The sampled $e(k)$, $\dot{\theta}^S(k)$, $a_x^S(k)$, $\theta^S(k)$, $v^S(k)$, and $d^S(k)$ may be defined similarly. When reformulating Eq. (10) in discrete time, the simulator state x_d in the cost J is factored in constraints instead of cost so that the resulting solution to the optimization problem is ensured to keep the simulator state within its constraints. In terms of the sampled data, Eq. (10) becomes

Find $u_s(k)$ that minimizes the cost J given by

$$J=\sum_{k=0}^{N}e^{\mathrm{T}}(k)\mathbf{Q}e(k)+u_s^{\mathrm{T}}(k)\mathbf{R}u_s(k)$$

subject to $\left|\dot{\theta}^s(k)\right|\le\dot{\theta}^s_{\max}$, $\left|a_x^s(k)\right|\le a^s_{x,\max}$, $\left|\theta^s(k)\right|\le\theta^s_{\max}$, $\left|v^s(k)\right|\le v^s_{\max}$, and $\left|d^s(k)\right|\le d^s_{\max}$. (11)

The main goal of this section is to turn Eq. (11) into a standard QP such as Eq. (16). Since $e, \dot{\theta}^S, a_x^S, \theta^S, v^S$ and d^S may be expressed in terms of x_e and x_d in Eqs. (8) and (9), These equations are converted into discrete time in order to obtain discrete-time dynamical models for $e(k), \dot{\theta}^S(k), a_x^S(k), \theta^S(k), v^S(k)$ and $d^S(k)$. The input to the system is $u_s(k)$, while $u_a(k)$ is assumed to be known *a priori*. The discrete-time vestibular model is obtained from Eq. (8) with the sampling period t_s, using a commercial tool, *e.g.*, MATLAB command c2d.m [14]. The resulting state equation for the sensation error in discrete time becomes

$$\begin{aligned}x_e(k+1)&=\mathbf{A}^d x_e(k)+\mathbf{B}^d u_s(k)-\mathbf{B}^d u_a(k)\\ e(k)&=\mathbf{C}^d x_e(k)+\mathbf{D}^d u_s(k)-\mathbf{D}^d u_a(k).\end{aligned}\tag{12}$$

Then, stack $u_a(k)$, $u_s(k)$ and $e(k)$ from 0th sample to Nth sample to obtain $\tilde{e}, \tilde{u}_a$ and $\tilde{u}_s$ $(\in\mathbb{R}^{2(N+1)})$

where $\tilde{e}=\begin{bmatrix}e(0)\\e(1)\\\vdots\\e(N)\end{bmatrix}, \tilde{u}_a=\begin{bmatrix}u_a(0)\\u_a(1)\\\vdots\\u_a(N)\end{bmatrix}, \tilde{u}_s=\begin{bmatrix}u_s(0)\\u_s(1)\\\vdots\\u_s(N)\end{bmatrix}$ and '$\sim$' is used to denote a stack of vectors throughout this paper. Solving Eq. (12) in discrete time gives an expression for $\tilde{e}$:

$$\tilde{e}\ =\ \mathbf{K}(\tilde{u}_s-\tilde{u}_a)\tag{13}$$

where $\mathbf{K}=\begin{bmatrix}\mathbf{D}^d & 0 & \dots & 0\\ \mathbf{C}^d\mathbf{A}^d\mathbf{B}^d & \mathbf{D}^d & \dots & 0\\ \vdots & \vdots & \ddots & \vdots\\ \mathbf{C}^d(\mathbf{A}^d)^{N-1}\mathbf{B}^d & \dots & \dots & \mathbf{D}^d\end{bmatrix}$.

The cost J is consequently represented in a quadratic form:

$$\begin{aligned}J\ &=\ \tilde{e}^T\tilde{\mathbf{Q}}_e\tilde{e}+\tilde{u}_s^T\tilde{\mathbf{R}}_s\tilde{u}_s\\ &=\ \tilde{u}_s^T\tilde{\mathbf{A}}\tilde{u}_s+\tilde{\mathbf{B}}\tilde{u}_s+\tilde{\mathbf{C}}\end{aligned}\tag{14}$$

where $\tilde{\mathbf{Q}}_e=diag[\overbrace{\mathbf{Q},\cdots,\mathbf{Q}}^{N+1}]$, $\quad\mathbf{Q}\in\mathbb{R}^{2\times2}$,

$\tilde{\mathbf{R}}_S=diag[\overbrace{\mathbf{R},\ \cdots\ ,\mathbf{R}}^{N+1}]$, $\quad\mathbf{R}\in\mathbb{R}^{2\times2}$,

$\tilde{\mathbf{A}} = \mathbf{K}^T\tilde{\mathbf{Q}}_e\mathbf{K} + \tilde{\mathbf{R}}_s$, $\tilde{\mathbf{B}} = -2\tilde{u}_a{}^T\mathbf{K}^T\tilde{\mathbf{Q}}_e\mathbf{K}$, $\tilde{\mathbf{C}} = \tilde{u}_a{}^T\mathbf{K}^T\mathbf{K}\tilde{u}_a$ and $\mathbf{Q}, \mathbf{R}$ are the weighting matrices.

Now, turn to the constraints. The state equation for the simulator state (4) may be solved to yield [15]

$$\begin{aligned}&\theta^s(k) = \theta^s(k-1) + \dot{\theta}^s(k-1)t_s,\\ &v^s(k) = v^s(k-1) + a_x^s(k-1)t_s,\\ &d^s(k+1) = d^s(k) + v^s(k)t_s + \frac{1}{2}a_x^s(k)t_s{}^2\\ &\text{for} \quad k = 0,1,\cdots,N+1.\end{aligned}$$

Since $\theta^S(k), v^S(k), d^S(k)$ (not to mention $\dot{\theta}^S(k)$ and $a_x^s(k)$) are given as summations of $\dot{\theta}^s$ and a_x^s and double summation of a_x^s, all of the constraints in Eq. (11) may be expressed in terms of $\tilde{u}_s$. The resulting constraint equations in Eq. (11) may be turned into a matrix inequality on $\tilde{u}_s$:

$$\mathbf{F}\tilde{u}_s \le \tilde{g} \tag{15}$$

$$\mathbf{F} = \begin{bmatrix}\mathbf{I}\\ -\mathbf{I}\\ \mathbf{M}\\ -\mathbf{M}\\ \mathbf{\Lambda N}\end{bmatrix}, \tilde{g} = \begin{bmatrix}g_1\\ g_1\\ g_2\\ g_2\\ g_3\end{bmatrix}, \mathbf{M} = t_s\begin{bmatrix}0 & 0 & 0 & \cdots & 0\\ I & 0 & 0 & \cdots & 0\\ I & I & 0 & \cdots & \vdots\\ \vdots & \vdots & \ddots & \ddots & \vdots\\ I & \cdots & I & I & 0\end{bmatrix},$$

$$\mathbf{N} = \begin{bmatrix}0 & 1 & 0 & 0 & \cdots & 0 & 0\\ 0 & 0 & 0 & 1 & \cdots & 0 & 0\\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots\\ 0 & 0 & 0 & 0 & \cdots & 0 & 1\\ 0 & -1 & 0 & 0 & \cdots & 0 & 0\\ 0 & 0 & 0 & -1 & \cdots & 0 & 0\\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots\\ 0 & 0 & 0 & 0 & \cdots & 0 & -1\end{bmatrix}, \mathbf{\Lambda} = \begin{bmatrix}\mathbf{L} & \mathbf{0}\\ \mathbf{0} & \mathbf{L}\end{bmatrix},$$

$$\mathbf{L} = \frac{t_s^2}{2}\begin{bmatrix}1 & 0 & \cdots & & 0\\ 3 & 1 & 0 & \cdots & 0\\ 5 & 3 & 1 & 0 & \vdots\\ \vdots & \vdots & \ddots & \ddots & \\ & & & & \vdots\\ (2N-1) & (2N-3) & & \cdots & 1\end{bmatrix},$$

$$g_1 = \begin{bmatrix}\dot{\theta}_{\max}\\ a_{\max}\\ \vdots\\ \dot{\theta}_{\max}\\ a_{\max}\end{bmatrix}, g_2 = \begin{bmatrix}\theta_{\max}\\ v_{\max}\\ \vdots\\ \theta_{\max}\\ v_{\max}\end{bmatrix} \text{ and } g_3 = \begin{bmatrix}d_{\max}\\ \vdots\\ \\ \vdots\\ d_{\max}\end{bmatrix}.$$

where $\mathbf{I}, \mathbf{M}, \mathbf{N}$ $(\in \mathbb{R}^{2(N+1)\times 2(N+1)})$, $\mathbf{L} \in \mathbb{R}^{(N+1)\times(N+1)}$, g_1, g_2 and g_3 $(\in \mathbb{R}^{2(N+1)})$.

Now, with Eq. (14) and Eq. (15), Eq. (11) may be turned into a standard QP which may be efficiently solved using standard QP solution mthodologies, e.g., the conjugate gradient (CG) algorithm [11]. The CG algorithm is chosen in this paper by virtue of its fast convergence and numerical efficiency. Eq. (11) is formulated into a QP:

$$\begin{aligned}&\underset{\tilde{u}_s \in R^{2(N+1)\times 1}}{\text{minimize}} \ q(\tilde{u}_s) = \frac{1}{2}\tilde{u}_s^T\mathbf{H}\tilde{u}_s + c^T\tilde{u}_s\\ &\text{subject to } \mathbf{F}\tilde{u}_s \le \tilde{g},\end{aligned} \tag{16}$$

where $\mathbf{H} = \mathbf{K}^T\tilde{\mathbf{Q}}_e\mathbf{K} + \tilde{\mathbf{R}}_s$ and $\mathcal{C} = -2\tilde{u}_a{}^T\mathbf{K}^T\tilde{\mathbf{Q}}_e\mathbf{K}$.

The following steps summarize the CG algorithm to solve a constrained QP Eq. (17), given $\mathbf{H}, c, \mathbf{F}$ and $\tilde{g}$.

Step 1: Set $i = 0$; select the initial point $\tilde{u}^{(0)} = 0$, the residual ς and tolerance τ.

Step 2: Calculate $g^{(0)} = \nabla q(\tilde{u}_s^{(0)})$. If $\left\|g^{(0)}\right\| \le \varsigma$, stop, else set $d^{(0)} = -g^{(0)}$.

Step 3: Calculate $\alpha_k = -\dfrac{g^{(i)T}d^{(i)}}{d^{(i)}\mathbf{H}d^{(i)}}$.

Step 4: $u_s^{(i+1)} = u_s^{(i)} + \alpha_k d^{(i)}$.

Step 5: $g^{(i+1)} = \nabla q(\tilde{u}_s^{(i+1)})$. If $\left\|g^{(i+1)}\right\| \le \varsigma$ and $\left\|\mathbf{F}\tilde{u}_s^{(i+1)} - \tilde{g}\right\| \le \tau$, stop. Otherwise,

Step 6: $\beta_k = \dfrac{g^{(i+1)T}\mathbf{H}d^{(i)}}{d^{(i)}\mathbf{H}d^{(i)}}$.

Step 7: $d^{(i+1)} = -g^{(i+1)} + \beta_k d^{(i)}$

Step 8: Set $i = i+1$; go to step 3.

The computational burden in solving a constrained QP with the aforementioned CG algorithm is now assessed. $\mathbf{H}, \mathcal{C}, \mathbf{F}$ are given as inputs to the algorithm. First, it takes $O(N^3)$ and $O(N^2)$ flops to compute $\mathbf{H} = \mathbf{K}^T\tilde{\mathbf{Q}}_e\mathbf{K} + \tilde{\mathbf{R}}_s$ and c, respectively. Computing $\mathbf{F}$ does not involve major computation since even unwieldy $\mathbf{\Lambda N}$ may be obtained via indexing instead of computing. The cost in step 2 is $O(N^2)$ flops to compute the gradient ($= \mathbf{H}\tilde{u}_s^{(0)} + \mathcal{C}$)

since step 2 involves a matrix-vector product ($O(N^2)$) and vector-vector summation ($O(N)$). In step 3, the required number of operations is $O(N^2)$ flops; $O(N)$ flops to calculate $g^{(i)T}d^{(i)}$ and $O(N^2)$ flops to compute $d^{(i)}\mathbf{H}d^{(i)}$. Step 4 costs $O(N)$ flops since it is only vector addition. The cost in step 5 is $O(N^2)$ flops to calculate $\nabla q(\tilde{u}_s^{(i+1)})$ and $\mathbf{F}\tilde{u}_s^{(i+1)}$. Step 6 requires $O(N^2)$ flops to compute the matrix-vector product as step 3. Finally step 7 takes $O(N)$ flops. For typical examples, the number of total iterations in solving Eq. (17) turns out to be 10~20. In the view of storage requirement, $\mathbf{H}$ and $\mathbf{F} \in \mathbb{R}^{5(N+1)\times 2(N+1)}$ need $O(N^2)$ storage space and vectors $u_s^{(i)}, g^{(i)}$ and $d^{(i)}$ require $O(N)$ bytes. In summary, the CG-based solution to Eq. (17) costs $O(N^3)$ flops and $O(N^2)$ storage space. This computational load is quite burdensome despite now readily available computing power since a typical ride in a motion simulator requires $N = 10^4 \sim 10^5$.

3.2 Fast algorithms for optimal trajectory design

In the standard solution described in Section 3.1, the tremendous computational load incurs since the matrix-matrix/matrix-vector products are performed without exploiting the structures of the underlying matrices. However, the QP problem Eq. (17) has matrices with special structures, which are either Toeplitz or Toeplitz-derived matrices. It is well-known that the Toeplitz matrix-vector products can be computed using Fast Fourier Transform (FFT) [16, 17]. Then, it is worth examining each step to see how it may be sped up.

First, consider the Hessian $\mathbf{H} = 2(\mathbf{K}^T\tilde{\mathbf{Q}}_e\mathbf{K} + \tilde{\mathbf{R}}_s)$, which needs to be pre-computed in the CG-based QP solver in Section 3.1. As previously shown, it takes $O(N^3)$ to compute $\mathbf{H}$. However, careful examination of the CG algorithm in the Section 3.1 indicates that $\mathbf{H}$ should not be pre-computed. Since $\mathbf{H}$ is used in matrix-vector products only (see steps 2, 3, 5 and 6 in Section 3.1), it is better to store its element matrices $\mathbf{K}, \tilde{\mathbf{Q}}_e$ and $\tilde{\mathbf{R}}_s$ where $\mathbf{K}$ is block-Toeplitz and $\tilde{\mathbf{Q}}_e, \tilde{\mathbf{R}}_s$ are block-diagonal. Then when each of the matrix-vector products involving $\mathbf{H}$ is computed, the element matrix-vector product is repeatedly computed, which costs only $O(N^2)$ flops. Moreover, the structures of the element matrices in $\mathbf{H}$ allow further reduction in computation as shown below. Now steps 2-6 are investigated in detail. Consider $\mathbf{K}\tilde{u}_s^{(i)}$, a sub-step in executing steps 2 and 5 that require $\mathbf{H}\tilde{u}_s^{(i)}$.

$$\mathbf{K}\tilde{u}_s^{(i)} = \begin{bmatrix} \mathbf{D}^d & 0 & \dots & 0 \\ \mathbf{C}^d\mathbf{B}^d & \mathbf{D}^d & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{C}^d(\mathbf{A}^d)^{N-1}\mathbf{B}^d & \dots & \dots & \mathbf{D}^d \end{bmatrix} \begin{bmatrix} u_s^{(i)}(0) \\ u_s^{(i)}(1) \\ \vdots \\ u_s^{(i)}(N) \end{bmatrix}$$

$$= \begin{bmatrix} \mathbf{D}^d u_s^{(i)}(0) \\ \mathbf{C}^d\mathbf{B}^d u_s^{(i)}(0) + \mathbf{D}^d u_s^{(i)}(1) \\ \vdots \\ \mathbf{C}^d(\mathbf{A}^d)^{N-1}\mathbf{B}^d u_s^{(i)}(0) + \dots + \mathbf{D}^d u_s^{(i)}(N) \end{bmatrix} \triangleq \delta^{(i)}.$$

There are N+1 blocks in the first column of $\mathbf{K}$ and each bock has 4 elements. If the computation is performed without utilizing the underlying structure, it requires $O(N^2)$ flops as noted earlier. Introduce new notations for $\mathbf{D}^d$, $\mathbf{C}^d\mathbf{B}^d$, $\cdots$, $\mathbf{C}^d(\mathbf{A}^d)^{N-1}\mathbf{B}^d$ ($\in \mathbb{R}^{2\times 2}$) for simplicity:

$$\mathbf{D}^d = \begin{bmatrix} k_{11}^1 & k_{12}^1 \\ k_{21}^1 & k_{22}^1 \end{bmatrix}, \mathbf{C}^d\mathbf{B}^d = \begin{bmatrix} k_{11}^2 & k_{12}^2 \\ k_{21}^2 & k_{22}^2 \end{bmatrix}, \cdots,$$

$$\mathbf{C}^d(\mathbf{A}^d)^k\mathbf{B}^d = \begin{bmatrix} k_{11}^{k+2} & k_{12}^{k+2} \\ k_{21}^{k+2} & k_{22}^{k+2} \end{bmatrix}$$

for $k = 0, 1, \cdots, N-1$

In addition, decompose $\delta^{(i)}$ into two vectors $\delta_e^{(i)}$ ($\in \mathbb{R}^{N+1}$) and $\delta_o^{(i)}$ ($\in \mathbb{R}^{N+1}$) that consist of its even and odd elements, respectively, so that $\delta_e^{(i)} = \delta^{(i)}(2k)$ and $\delta_o^{(i)} = \delta^{(i)}(2k+1)$ for $k = 0, 1, \cdots, N$. $\tilde{u}_s^{(i)}$ may be factored into $\tilde{u}_{s,e}^{(i)}$ and $\tilde{u}_{s,o}^{(i)}$ in a similar manner. Then, it can be shown $\delta_e^{(i)}$ and $\delta_o^{(i)}$ may be expressed as summations of Toeplitz-vector products, respectively:

$$\delta_o^{(i)} = \mathbf{T}_{11}\delta_o^{(i)} + \mathbf{T}_{12}\delta_e^{(i)}, \delta_e^{(i)} = \mathbf{T}_{21}\delta_o^{(i)} + \mathbf{T}_{22}\delta_e^{(i)}$$

where

$$\mathbf{T}_{11}=\begin{bmatrix} k_{11}^1 & 0 & \cdots & 0 \\ k_{11}^2 & k_{11}^1 & 0 & \vdots \\ k_{11}^3 & k_{11}^2 & \ddots & \\ \vdots & & \ddots & \\ k_{11}^{N+1} & \cdots & & k_{11}^1 \end{bmatrix},\quad \mathbf{T}_{12}=\begin{bmatrix} k_{12}^1 & 0 & \cdots & 0 \\ k_{12}^2 & k_{12}^1 & 0 & \vdots \\ k_{12}^3 & k_{12}^2 & \ddots & \\ \vdots & & \ddots & \\ k_{12}^{N+1} & \cdots & & k_{12}^1 \end{bmatrix},$$

$$\mathbf{T}_{21}=\begin{bmatrix} k_{21}^1 & 0 & \cdots & 0 \\ k_{21}^2 & k_{21}^1 & 0 & \vdots \\ k_{21}^3 & k_{21}^2 & \ddots & \\ \vdots & & \ddots & \\ k_{21}^{N+1} & \cdots & & k_{21}^1 \end{bmatrix},\quad \mathbf{T}_{22}=\begin{bmatrix} k_{22}^1 & 0 & \cdots & 0 \\ k_{22}^2 & k_{22}^1 & 0 & \vdots \\ k_{22}^3 & k_{22}^2 & \ddots & \\ \vdots & & \ddots & \\ k_{22}^{N+1} & \cdots & & k_{22}^1 \end{bmatrix}$$

The above Toeplitz matrices can be embedded into the corresponding circulant matrices. Then, since the circulant matrices are diagonalized by the Fourier transformation matrices [16, 18], the matrix-vector product ($\mathbf{K}\tilde{u}_s^{(i)}$) may be carried out using FFT, which reduces the flop count to $O(N\log_2 N)$ [16]. Overall, it takes $O(N\log_2 N)$ flops to calculate $\mathbf{H}\tilde{u}_s$ since $\tilde{\mathbf{Q}}_e, \tilde{\mathbf{R}}_s$ are block-diagonal and $\mathbf{K}^T$ is also block-Toeplitz. The linear cost matrix ($\mathcal{C} = -2\tilde{u}_a{}^T\mathbf{K}^T\tilde{\mathbf{Q}}_e\mathbf{K}$) may be computed in the same way, so that the resulting cost of step 2 becomes $O(N\log_2 N)$

It turns out that the idea of speeding up $\mathbf{K}\tilde{u}_s^{(i)}$ computation plays a crucial role in speeding up the CG-based solution methodology in Section 3.1. Note that the same structure exists in computing $\mathbf{K}d^{(i)}$, a sub-step in executing steps 3 and 6 with $\tilde{u}_s^{(i)}$ substituted for $d^{(i)}$. As a result, steps 2-6 except for step 5 may be executed at $O(N\log_2 N)$ flops. Finally, consider step 5. Since $\mathbf{F}$ consists of sub-matrices $\mathbf{I},\mathbf{M},\mathbf{\Lambda N}$, the computation on each sub-matrix may be performed separately when calculating $\mathbf{F}\tilde{u}_s^{(i+1)}$. Since $\mathbf{N}$ is used to select even-positioned element, it does not involve any computational cost. Since $\mathbf{I}$, $\mathbf{M}$, $\mathbf{\Lambda}$ are block-Toeplitz and $\mathbf{L}$ is truly Toeplitz, FFT again reduces the total computational complexity of step 5 to $O(N\log_2 N)$ flops.

Now, turn to the storage requirement. First, consider the Hessian $\mathbf{H}$. Note that it takes $O(N^2)$ storage space to store $\mathbf{H}$. However, since only the matrix-vector products need to be computed and $\mathbf{K}$, $\tilde{\mathbf{Q}}_e$, $\tilde{\mathbf{R}}_s$ are block-Toeplitz and block diagonal, it is enough to store only the first block columns or the block diagonals of the element matrices. For $\mathbf{K}\in\mathbb{R}^{2(N+1)\times 2}$, only the following matrix $\mathbf{K}_T$ is stored:

$$\mathbf{K}_T = \begin{bmatrix} \mathbf{D}^d \\ \mathbf{C}^d\mathbf{B}^d \\ \vdots \\ \mathbf{C}^d(\mathbf{A}^d)^{N-1}\mathbf{B}^d \end{bmatrix},$$

which requires only $O(N)$ bytes. Even smaller storage space is needed for $\tilde{\mathbf{Q}}_e$ and $\tilde{\mathbf{R}}_s$ since they consist of $\mathbf{Q}$ and $\mathbf{R}$, both 2×2 matrices. Therefore, the whole required storage space in steps 2, 3, 5 and 6 is $O(N)$ bytes. In step 5, the storage space for $\mathbf{F}$ can be reduced to $O(N)$ bytes because $\mathbf{M},\mathbf{\Lambda}$, and $\mathbf{L}$ are block-Toeplitz and $\mathbf{I}$ is diagonal. Overall, the proposed fast algorithm requires only $O(N\log_2 N)$ flops and $O(N)$ storage space. In summary, Table 1 compares the computational loads of the existing algorithm in Section 3.1 and the fast algorithm in Section 3.2.

In a typical ride that spans 300 seconds at the sampling rate 100Hz, N becomes 30000 and the required flop counts are on the order of 10^5 in the proposed fast algorithm. Flops counts are drastically reduced compared to those in the existing algorithm, which is on the order of 10^{13}. Since typically it takes eight bytes to store a double precision real number by IEEE, floating-point standard [10], the required storage in the fast algorithm is on the order of mega-bytes instead of giga-bytes in the existing algorithm. Such reduction in flop counts and storage space allows generating optimal trajectories for extremely long rides, *e.g.*, $N = 10^8$.

Table 1. Comparison of the computational loads for the existing and and proposed algorithms.

	Flop counts		Storage space(bytes)	
Steps	Existing	Proposed	Existing	Proposed
Pre-step	$O(N^3)$	Unnecessary	$O(N^2)$	Unnecessary
2	$O(N^2)$	$O(N\log_2 N)$	$O(N^2)$	$O(N)$
3				
5	$O(N^2)$	$O(N\log_2 N)$	$O(N^2)$	$O(N)$
6				
Total	$O(N^3)$	$O(N\log_2 N)$	$O(N^2)$	$O(N)$

Before proceeding, it must be noted that the proposed QP-based trajectory design methodology is primarily applicable to the cases with the trajectory given *a priori* and has limitations when the interactive ride is desired. Yet, with the advent of the numerically efficient implementation developed in this subsection, the QP-based trajectory design methodology may be used to generate the filtered trajectory even without a predetermined trajectory. By assuming a finite window (horizon), a pseudo-optimal trajectory may be obtained over the finite window. Whenever the trajectory is changed by the user (or rider), the filtered trajectory is re-computed and updated. Although not optimal in any sense, the resulting filtered trajectory has potential to utilize the simulator constraints to great extent.

4. Performance analasis on the Eclipse-II motion simulator

4.1 Eclipse-II motion simulator

A motion simulator consists of an audio system to generate sounds, a visual system to display images, and a motion base system to generate movement according to motion cues. Most conventional simulators adopt the Stewart platform as the motion base [18-20]. The Stewart platform is a six degrees-of-freedom parallel mechanism that enables both translational and rotational motions [9, 20]. However, such motions as the 360-degree overturn are impossible in the Stewart platform, because the platform can only tilt as much as ± 20-30 degrees. That is, it cannot reproduce the overturn motion of the aircraft or the 360-degree spin of the roller coaster. A novel six degrees-of-freedom parallel mechanism architecture, which is called the Eclipse-II, has been designed for the motion base of a motion simulator [9, 10]. Fig. 2 shows the Eclipse-II mechanism and an example of its rotational motion capability [9]. Since the Eclipse-II has no limitation on its rotational motion, it is possible to develop a more realistic and higher fidelity simulator by adopting the Eclipse-II mechanism as the motion base of a flight simulator or a roller coaster motion simulator.

Despite these merits, the Eclipse-II motion simulator has a small workspace especially in its longitudinal motion, which limits its capability of reproducing the specific force sensation. In this respect, the Eclipse-II motion simulator brings up quite a challenging task of optimal-trajectory design. In the following subsection, the novel trajectory design approach proposed in Section 3 is applied to generate optimal trajectories for the Eclipse-II motion simulator.

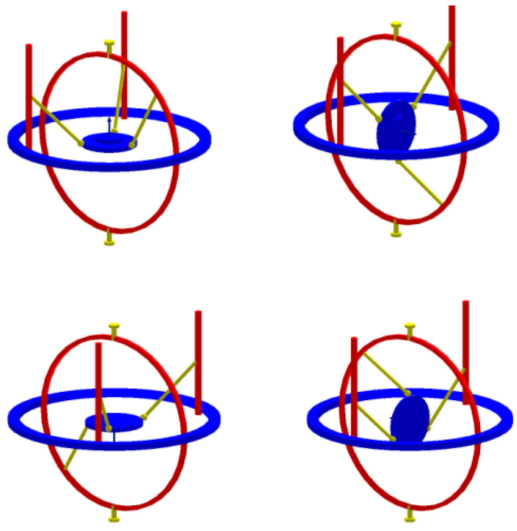
Fig. 2. Eclipse-II mechanism and its 360-degree continuous rotational motions.

4.2 Performance analysis

The major contributions of this paper are two-fold:

1. A novel approach is proposed to generate optimal trajectories for a motion simulator based on the constrained QP.
2. A fast algorithm is developed to solve the constrained QP by exploiting the Toeplitz structures of the underlying matrices.

The performance analysis in this section focuses on the first item after mentioning that the fast algorithm in Section 3.2 is verified to be equivalent to the standard one in Section 3.1 in terms of resulting solutions. Therefore, a distinction between two algorithms is not made throughout this section. The performance of the proposed algorithm is then benchmarked against only the LQR-based washout algorithm by Telban and Cardullo [8] among others, noting that the LQR-based washout algorithm outperforms other existing ones by generating filtered trajectories in the most systematic and accountable manner. Since it is the explicit formulation of the simulator state x_d as constraints into the optimization problem that gives the edge to the QP-based algorithm over the LQR-based one, special attention must be paid to how the two algorithms handle the constraints on x_d as well as the sensation error when comparing performance.

During the performance analysis, both algorithms use the same values for physical parameters in Eq. (1) and Eq. (3) and the translational break frequency unless mentioned otherwise [8, 12]. The spatial and dynamical constraints on the Eclipse- II motion simulator are given as

● the radius of the workspace cylinder	37.2 mm
● the height of the workspace cylinder	78.6 mm
● maximum linear velocity	0.2 m/s
● maximum linear acceleration	0.1g or 0.98m/s^2
● maximum angular rate	2 rad/s

where the workspace of Eclipse-II is described as a cylinder for convenience [11, 21]. It must be noted that the version of the Eclipse-II motion simulation under consideration is a working sample of the full-fledged one (built for proof of concept) and has poor spatial and dynamical capabilities for a motion simulator. Ironically, such limited capabilities make the problem of designing optimal trajectories more challenging and the performance gap between different algorithms more pronounced. Another point worth mentioning in regard to the constraints is that the workspace of the working sample has relatively smaller lateral span than the vertical one.

The simulation studies are primarily conducted on the longitudinal response of angular velocity and linear acceleration (or specific force) to the pitch/ surge motions (extensions to other motions are rather trivial). Numerous trajectories have been used to compare the performances of the LQR- and QP-based algorithms. Despite comparable performances in a few occasions, the QP-based algorithm consistently show superior performance in most trajectories. Whenever a trajectory requires the simulator to push the limits of its capabilities, the performance gap between the two algorithms becomes obvious. This paper presents detailed examinations on three representative trajectories for terseness and clarity: doublet pitch, linear acceleration, and simultaneous application of these two input trajectories. The three input trajectories span over 4 seconds. The sample rate of the trajectories is chosen to be 100Hz to allow smooth transition from sample to sample. For each input trajectory, the LQR- and QP-based algorithms compute the optimized angular velocity ($\dot{\theta}^s$) and linear acceleration (a_x^s). Since the sensation error is of utmost interest, the raw data $\dot{\theta}$ and a_x are processed with the vestibular model Eq. (1) and Eq. (3) to yield the sensed angular velocity $\hat{q}$ and specific force $\hat{a}_x$. It is worth noting that LQR-based algorithm requires repeated weight ($\mathbf{Q}$, $\mathbf{R}$, and $\mathbf{R}_d$) tuning in order to generate the results presented here while the QP-based one produces the results in one-shot. It should be noted that all conditions on the simulation are intended to validate the capability of the proposed QP-based approach to fully utilize the workspace of the simulator and that they exceed the constraints on the simulator.

First, a doublet pitch is examined as an input trajectory, which has the maximum magnitude of 3 rad/s. Therefore, the corresponding input force consists of only angular velocity term without any linear acceleration as shown in Fig. 3 (a) and (b), respectively. Note that the maximum magnitude exceeds the simulator constraint on the angular rate 2 rad/s. Fig. 3 (a) shows that the input angular velocity to the simulator goes off its constraint between 1 and 2 seconds. Both the LQR- and QP-based algorithms satisfy the constraint by keeping the optimized angular velocity below 2 rad/s, as expected. Other than that, the optimized trajectories of angular velocities and linear accelerations from the two algorithms differ qualitatively to great extent. The LQR-based algorithm simply reduces the magnitude of the simulator states, which results in the scale-down angular velocity profile. On the other hand, the QP-based algorithm takes full advantage of the allowable range of the angular velocity by 1) reproducing the angular velocity until it saturates, 2) keeping it at the maximum once it saturates 3) shaking the induced linear acceleration (Fig. 3 (b)) simultaneously to reject the effect of the saturated $\dot{\theta}^s$. As a result, the sensed angular velocity $\hat{q}$ and sensed specific force $\hat{a}_x$ from the QP-based algorithm track the corresponding $\hat{q}$ and $\hat{a}_x$ from the input trajectory much better than the ones from the LQR-based algorithm as shown in Figs. 3 (c) and (d). The QP-based algorithm truly minimizes the sensation error better although the LQR-based one produces the sensed $\hat{q}$ and $\hat{a}_x$ whose shapes only resemble the ones from the input trajectory closely. It is noteworthy that the better-coordinated effort between the rotational and translational motions has helped to further reduce the sensation error in the QP-based approach. In other words, although the apparent

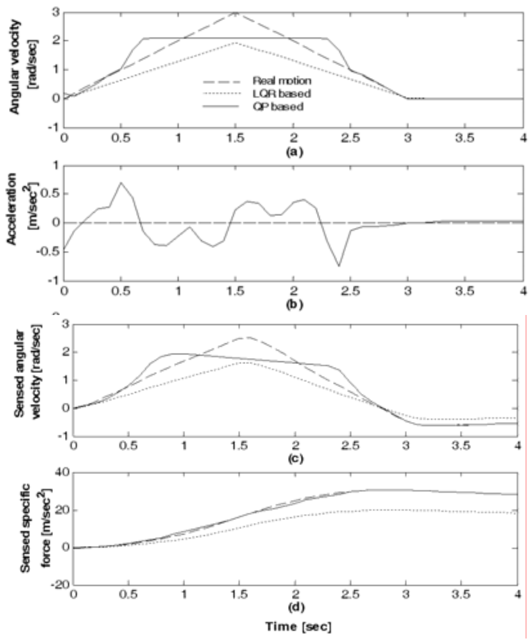

Fig. 3. Comparison of LQR- and QP-based trajectories to pitch input; (a) angular velocities, (b) linear accelerations, (c) sensed angular velocities, (d) sensed specific forces.

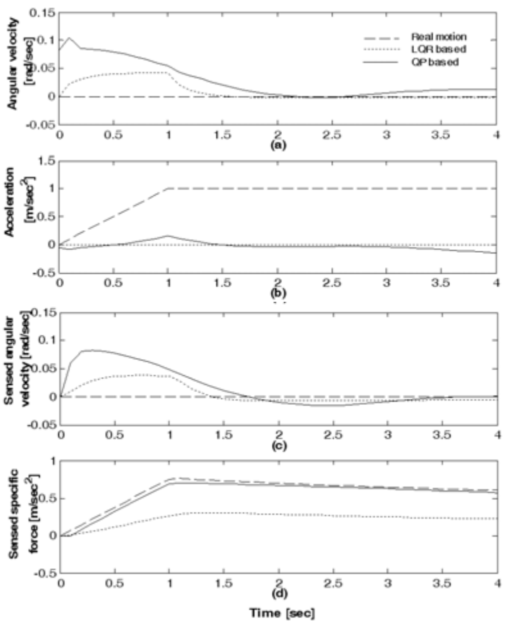

Fig. 4. Comparison of LQR- and QP-based trajectories to surge input; (a) angular velocities, (b) linear accelerations, (c) sensed angular velocities, (d) sensed specific forces.

pattern of the linear acceleration in Fig. 3 (b) does not resemble that of the real motion, the rotational motion effectively compensates for the discrepancy in the linear motion by virtue of the coupling between the two motions (as manifested by Eq. (4)). Overall, the explicit formulation of the simulator state x_d as constraints provides clear performance improvement in the QP-based approach over the LQR-based counterpart. Moreover, the QP-based approach achieves such performance improvement without iterative weight tuning (necessary in the LQR-based approach in order to keep x_d within the simulator constraints).

Next, consider the linear acceleration as an input trajectory without angular velocity shown in Fig. 4 (a) and (b). The linear acceleration is increased to 1.0 m/s^2 till 3 seconds and kept at the same level afterward. Considering that the simulator constraint on linear acceleration is 0.98 m/s^2, the current input trajectory also pushes the simulator to exceed its constraints. Recall that the working sample of the Eclipse-II motion simulator has a severe limitation on translation along the longitudinal direction in its workspace, which makes it impossible to directly realize the linear acceleration with high fidelity. Such an apparent impasse is circumvented by judicious utilization of the human vestibular system, *i.e.,* creating the sensed specific force by tilting the motion platform. Figs. 4 (a) and (b) manifest the role of the crossover path, where the rotational motion helps to realize the translational motion in both LQR- and QP-based approaches. Note that the level of the angular velocity is very low in Fig. 4 (a), which prevents the pilot or rider from experiencing motion miscue, *i.e.,* rotational motion instead of translation motion. Also, note that both LQR- and QP-based approaches do not push the simulator to its linear acceleration limit due mainly to its workspace constraints. In other words, what determines the limit is not acceleration, but displacement. Although Fig. 4 (c) displays a slight motion miscue in terms of sensed angular velocity (special attention must be paid to the small magnitude along the y-axis in Figs. 4 (a) and (c), Fig. 4 (d) supports that the gain in sensed specific force outweighs the negligible motion miscue. It is clear that the QP-based approach outperforms the LQR-based one as far as the sensation error is concerned. Again, the QP-based approach reproduces the sensed specific force too close to distinguish from the one from the input trajectory while the LQR-based one is content with a trajectory that resembles in shape but leads to significant sensation error.

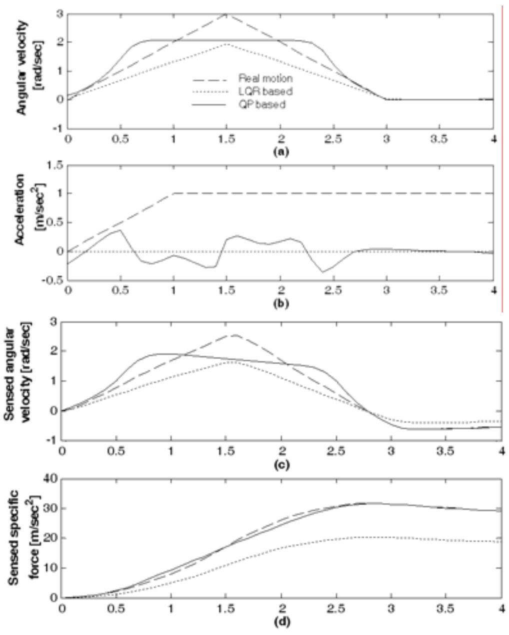

Fig. 5. Comparison of LQR- and QP-based trajectories to pitch input/ surge input; (a) angular velocities, (b) linear accelerations, (c) sensed angular velocities, (d) sensed specific forces.

Finally, the doublet pitch in Fig. 3 (a) and the linear acceleration in Fig. 4 (a) are exerted simultaneously as an input trajectory. Similar interpretations from the previous two cases may be made in Fig. 5. The QP-based approach generates a trajectory that results in much smaller sensation errors in the sensed angular velocity and specific force. In other words, even with the combined motion as an input trajectory, the trend in the previous two cases continues. The performance analysis presented in this section clearly demonstrates that the proposed QP-based algorithm is capable of generating optimal trajectories in a systematic, accountable and intuitive manner, not to mention its blazingly fast implementation.

5. Concluding remarks

A novel methodology of generating an optimal trajectory for a motion simulator is developed that explicitly takes into account, and thereby even exploits, the simulator constraints. Building upon the existing approach based on the LQR theory, the proposed algorithm tries to minimize the human sensation error while ensuring the simulator stays within its constraints. It turns out that when the trajectory is given *a priori*, the problem of computing the optimal trajectory can be recast into a constrained QP. Although it may be readily solved by using now commercially available tools when the trajectory is not too long, the constrained QP calls for a fast algorithm that runs faster with less storage space. By taking advantage of the Toeplitz structures of the underlying matrices, an orders-of-magnitude faster algorithm is obtained that requires much less storage space. The viability of the proposed algorithm is tested on the Eclipse-II motion simulator. The simulation results show that the proposed QP-based algorithm outperforms the existing LQR-based one mainly in three accounts: small sensation error, full utilization of the workspace and machine capacity, and no need for iterative weight-tuning. The proposed algorithm has been successfully used to generate optimal trajectories for the Eclipse-II motion simulator at Seoul National University in Korea. Overall, the following practical scheme is proposed while weighing the performance and complexity together: in normal situations, the conventional LQR-based approach should be taken by virtue of its simplicity and reasonable performance, and in other situations where certain trajectories violate the motion limit of the simulator, the LQR-based approach should be switched to the QP-based one capable of effectively handling severe constraints as shown in this paper.

References

[1] F. Barbagli, D. Ferrazin, C.A. Avizzano, M. Bergamasco, Washout filter design for a motorcycle simulator, Proc. of the IEEE Virtual Reality, Yokohama, Japan. (2001) 225-232.

[2] I. Rock, An introduction to perception, Macmillan, New York, USA, (1975) Chap. 5.

[3] L. R. Young, Visual vestibular interaction, sixth international symposium on biocybernetics, control mechanisms in bio-and ecosystems, International Federation of Automatic Control, Leipzig, East Germany. (1977).

[4] W. H. Levinson, and A. M. Junker, A model for the pilot's use of motion cues in steady-state roll axis tracking tasks, Proc. of the AIAA Flight Simulation Technology, Arlington, Texas, USA. (1978) 190-197.

[5] L. D. Reid and M. A. Nahon, Flight simulation motion-base drive algorithms: Part 1 – Developing and testing the equations, *UTIAS Report*. No. 296 (1985) CN ISSN 0082-5255.

[6] W. Wu and F. M. Cardullo, Is there an optimum cueing algorithm? AIAA Modeling and Simulation

Technologies Conference, New Orleans, LA. (1997) 23-29.

[7] M. Nahon and D.L. Reid, Adaptive simulator motion software with supervisory control, *Journal of Guidance, Control and Dynamics 15*. 15 (2) (1992) 376-383.

[8] F. M. Cardullo and R. J. Telban, Development in human centered cueing algorithms for control of flight simulator motion systems, Proc. of the AIAA Modeling and Simulation Technologies Conference, AIAA-99-4328. (1999).

[9] J. Hwang, Analysis and design of the eclipse- II parallel mechanism for motion simulator, Ph.D. thesis, School of Mechanical and Aerospace Engineering, Seoul National University, Korea. (2002).

[10] J. W. Kim, J. C. Whang, J. C. Kim, F. C. Park, Eclipse-II: A new parallel mechanism enabling continuous 360-degree spinning plus three-axis translational motions, *IEEE Transactions on Robotics and Automation*. 18 (3) (2002) 367-373.

[11] Edwin K. P. Chong and H. Z. Stanislaw, An introduction to optimization, John Wiley & Sons, Inc. (1996).

[12] R. J. Telban, F. M. Cardullo and L. Guo, Investigation of mathematical models of otolith organs for human centered motion cueing algorithm, Proc. of the AIAA Modeling and Simulation Technologies Conference, Denver, Colorado, AIAA 99-4328. (2000).

[13] T. Kailath, Linear system, Englewood Cliffs, Prentice-Hall, Inc. (1980).

[14] D. Hanselman and B. Littlefield, Mastering MATLAB 6, Prentice-Hall, Inc. (2000).

[15] M. O. James, Numerical analysis, SIAM, Philadelphia. USA, (2002).

[16] P. J. Davis, Circulant matrices, John Wiley & Sons. (1979).

[17] H. G. Gene and F. Van Loan. Charles, Matrix computations, Johns Hopkins, USA, (1996).

[18] J. B. Song, U. J. Jung and H. D. Ko, Washout algorithm with fuzzy-based tuning for a motion simulator, *KSME I. J*. 17 (2) (2003) 221-229.

[19] K. S. You, M. C. Lee, E. Kang and W.S. Yoo, Development of a washout algorithm for a vehicle driving simulator using new tilt coordination and return mode, *KSME I.J.* 19 (1) (2005) 272-282.

[20] D. Li and S. E. Salcudean, Modeling, Simulation, and control of a hydraulic stewart platform, Proc. of the IEEE International Conference on Robotics and Automation, Albuquerque, New Mexico. (1997).

[21] J. L. Meiry, The vestibular system and human dynamic space orientation, NASA CR-628 (1966).