Model Predictive Driving Simulator Motion Cueing Algorithm with Actuator-based Constraints

*Nikhil J.I. Garrett[a] and Dr Matthew C. Best[a]

[a]Dept Aeronautical and Automotive Engineering, Stewart Miller Building, Loughborough University, Loughborough, Leicestershire, LE11 3TU, UK

Phone: +44 (0)1509 227264

Fax: +44 (0)1509 227275

E-mail: N.J.I.Garrett@lboro.ac.uk*, M.C.Best@lboro.ac.uk

* Corresponding author

Abstract

The simulator motion cueing problem has been considered extensively in the literature; approaches based on linear filtering and optimal control have been presented and shown to perform reasonably well. More recently, Model Predictive Control (MPC) has been considered as a variant on the optimal control approach; MPC is perhaps an obvious candidate for motion cueing due to its ability to deal with constraints, in this case the platform workspace boundary.

This paper presents an MPC-based cueing algorithm that, unlike other algorithms, uses the actuator positions and velocities as the constraints. The result is a cueing algorithm that can make better use of the platform workspace whilst ensuring that its bounds are never exceeded. The algorithm is shown to perform well against the classical cueing algorithm and an algorithm previously proposed by the authors in simulation and in tests with human drivers.

Keywords: Vehicle simulators; Predictive control; Model-based control; Driver behaviour

1 Introduction

The inclusion of motion in driving simulators has been shown to improve the perceived quality of simulation [1] and to prompt people to drive the simulated vehicle in a more natural way [2]. The most prevalent platform architecture is the six degrees-of-freedom (DOF) Stewart platform, as used on the Loughborough University driving simulator.

The actuators on the Loughborough simulator have a maximum stroke of 600mm, which corresponds to maximum excursion of the order ±0.5m and ±20° about the neutral position.

However, these values are for motion in a single axis; the geometry of the platform means that the workspace has the form shown in figure 1, so motion in multiple axes simultaneously results in reduced possible excursion in each.

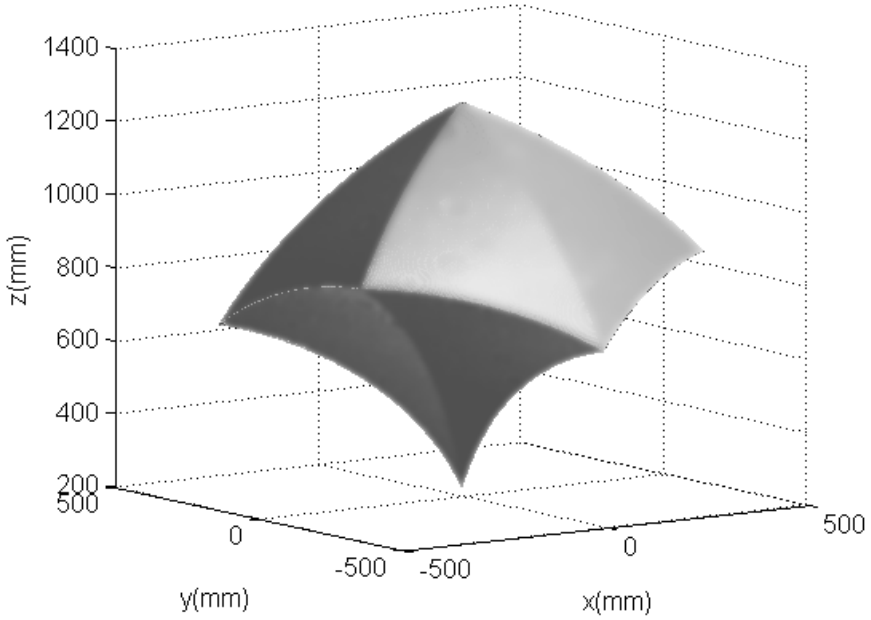

Figure 1 – Illustration of platform workspace envelope for motion in x,y,z directions

The limited platform workspace necessitates a transformation from the simulated vehicle motion to the achievable platform motion; this transformation is known as the motion cueing algorithm. The cueing algorithm should reproduce as much of the important motion as possible whilst remaining within the platform envelope at all times. In general existing cueing algorithms, including most of those described in this section, treat the six DOF separately and so do not optimize use of the available space envelope.

An early approach [3], now known as the classical algorithm, uses linear high-pass filters to remove the large-amplitude steady-state motion together with an appropriate gain, chosen to ensure platform limits are respected for the anticipated worst-case manoeuvre. A development of this algorithm [4] replaces the fixed gain with an adaptive gain, the effect of which is to reduce 'false cues', where the simulator motion is in the opposite direction to the vehicle motion.

The linear optimal control-based algorithm [5] defines the motion cueing as a tracking problem where the perceived simulator motion should track the perceived vehicle motion. A linear quadratic regulator (LQR) calculates the optimal feedback gains to minimise a cost function that penalises perception error, platform excursion and the control input; the relative weighting of these terms is tuned to give the desired behaviour.

Implementations of the cueing algorithms described above frequently include tilt coordination (see e.g. [3]), where roll and pitch motion are used to simulate low frequency lateral and longitudinal accelerations respectively, figure 2.

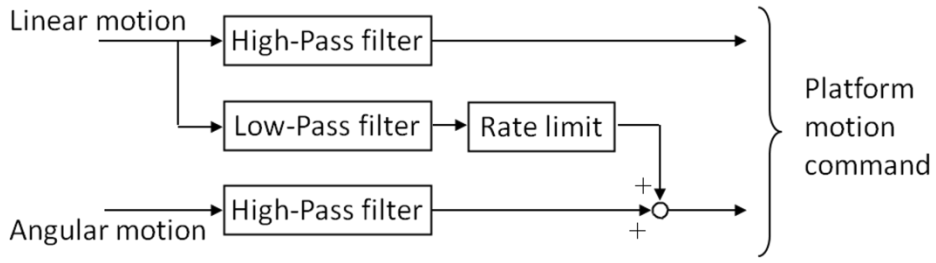

Figure 2 – Tilt coordination topology

These existing approaches make poor use of the available workspace; firstly, they must be tuned such that a worst-case motion respects the platform limits, so for anything other than this case only a fraction of the available excursion is used. Secondly, since the six DOF are considered separately, the excursion limits in each DOF must be reduced to take into account motion in other axes. A notable exception to the latter point is the actuator state-based variant of the adaptive algorithm proposed by Grant and Naseri [6]; this replaces the Cartesian cost terms in the adaptive cost function with actuator state terms. The result is a controller that can better deal with motion in multiple Cartesian DOFs.

The fact that Model Predictive Control (MPC) explicitly accounts for the controlled system's constraints makes it an interesting possibility for motion cueing. So far, two publications have proposed MPC motion cueing algorithms. The first, by Dagdelen et al. [7], is based on a modified perception tracking approach; over the prediction horizon H_p, the optimal control sequence is chosen that minimises the motion perception error for the first two time steps and then for the remaining H_p-2 steps returns to centre below the human perception threshold. The resulting controller matches the perceived accelerations until a platform limit is imminent, at which point the platform returns to centre; if the sign of the vehicle acceleration subsequently changes, perception matching resumes. An issue with MPC is the computational burden of doing an optimization at each time step; Dagdelen et al. [7] reduce the online computation time using the invariant set method of Gutman and Cwikel [8]. Their initial tests indicated that the transition between full acceleration rendering and the washout phase was too abrupt, and an artificial smoothing of the transition was added. It is reported that expert drivers tested the algorithm against the classical filtering method, and that the MPC algorithm was preferred, though no data or analysis is presented.

The second MPC motion cueing work is by Augusto [9]; this is a more 'traditional' approach in that the formulation is a tracking problem with cost on platform excursion and control input. The MPC problem is solved by calculating the associated Quadratic Programming (QP) problem and solving using a commercially-available solver. The controller produces much smoother motion commands than that proposed by Dagdelen et al. [7]. Some simulation results are presented that indicate that the MPC algorithm makes better use of the workspace than the classical algorithm. However, as yet no results with human test drivers appear to have been published.

This paper describes a new MPC cueing algorithm that is designed to be simple to implement and tune and which can fully exploit the platform workspace. The algorithm, described in

section 2, uses actuator lengths and velocities as the spatial constraints instead of Cartesian quantities, thus permitting full use of the platform workspace even with motion in several axes concurrently. In order to simplify the implementation, a technique for real-time optimization is described that precludes the need for any pre-calculation or approximation of the MPC problem; the solver, also in section 2, is easily implemented using straightforward linear algebra. In section 3, the new algorithm is evaluated in simulation and compared with the classical algorithm and another algorithm previously developed by the authors. It is also implemented on the Loughborough simulator, and a small study with human drivers is presented in section 4.

2 MPC algorithm derivation

Model Predictive Control uses receding horizon control; at each time step, a control sequence is calculated over a control horizon H_u that minimises a cost function over the prediction horizon H_p (with $H_u \leq H_p$). The first value in the control sequence is applied to the system, and the process is then repeated at the next time step. A linear model of the system is included to predict the effect of the control sequence on the system over the prediction horizon.

Given that this algorithm uses actuator states as the constrained platform variables it would seem sensible to create a six input-six output controller, as motion in each DOF has an effect on all six actuators. However, initial investigation indicated that the computational burden would be too high; instead, the six DOFs are divided as per most other cueing algorithms, into two tilt-coordinated pairs (longitudinal/pitch and lateral/roll) and two single-DOF cases (vertical and yaw). Clearly this will affect constraint handling to some extent; this point is addressed in section 2.1.

2.1 Linear system model

This section describes the formation of the lateral/roll controller; the other controllers are formed in the same way, with only the vestibular model and number of DOFs differing.

2.1.1 Vestibular dynamics

As with most optimal control-based algorithms, the MPC motion cueing is set up as a tracking problem; the perceived motion in the simulator tracks the perceived motion in the real vehicle whilst remaining within the platform constraints. Working with perceived motion requires a model of the human vestibular system; this comprises the otoliths (linear acceleration sensors) and the semicircular canals (angular velocity sensors). Various models are proposed in the literature for the vestibular dynamics; the linear models used by Augusto [9], which are based on the work by Reid and Nahon [10], are used here. The semi-circular canal transfer function from head angular velocity ω to perceived angular velocity $\hat{\omega}$ is

$$\frac{\hat{\omega}}{\omega} = \frac{\tau_a \tau_2 s}{(\tau_a s + 1)(\tau_1 + 1)(\tau_2 s + 1)} \tag{1}$$

The otolith transfer function from acceleration a to perceived acceleration $\hat{a}$ is

$$\frac{\hat{a}}{a} = \frac{K_{oto}(\tau_n s + 1)}{(\tau_S s + 1)(\tau_L s + 1)} \tag{2}$$

The perceived motion for the simulated vehicle is calculated externally and is the reference input to the controller; the perceived motion in the simulator is calculated as part of the controlled system's dynamics.

For the lateral/roll controller the perceived quantities are the roll velocity *p* and lateral acceleration a_y. First, the semi-circular canal transfer function is converted to discrete state-space form,

$$\boldsymbol{x_{scc}}(k+1) = \boldsymbol{A_{scc}}\boldsymbol{x_{scc}}(k) + \boldsymbol{B_{scc}}p(k)$$
$$\hat{p}(k) = \boldsymbol{C_{scc}}\boldsymbol{x_{scc}}(k)$$

$$\boldsymbol{A_{scc}} = \begin{bmatrix} -\frac{1}{\tau_a} - \frac{1}{\tau_1} - \frac{1}{\tau_2} & 1 & 0 \\ -\frac{1}{\tau_a \tau_1} - \frac{1}{\tau_a \tau_2} - \frac{1}{\tau_1 \tau_2} & 0 & 1 \\ -\frac{1}{\tau_a \tau_1 \tau_2} & 0 & 0 \end{bmatrix}, \boldsymbol{B_{scc}} = \begin{bmatrix} \frac{1}{\tau_1} \\ 0 \\ 0 \end{bmatrix}, \boldsymbol{C_{scc}} = [1 \quad 0 \quad 0] \tag{3}$$

The same is done for the otolith transfer function, with the addition of the contribution from tilt coordination:

$$\boldsymbol{x_{oto}}(k+1) = \boldsymbol{A_{oto}}\boldsymbol{x_{oto}}(k) + \boldsymbol{B_{oto}}\boldsymbol{a_y}(k) + \boldsymbol{B_{oto}}g\phi(k)$$
$$a_y(k) = \boldsymbol{C_{oto}}\boldsymbol{x_{oto}}(k)$$

$$\boldsymbol{A_{oto}} = \begin{bmatrix} -\frac{1}{\tau_L} - \frac{1}{\tau_S} & 1 \\ -\frac{1}{\tau_L \tau_S} & 0 \end{bmatrix}, \boldsymbol{B_{oto}} = \begin{bmatrix} \frac{K_{oto}\tau_n}{\tau_L \tau_S} \\ \frac{K_{oto}}{\tau_L \tau_S} \end{bmatrix}, \boldsymbol{C_{oto}} = [1 \quad 0] \tag{4}$$

Note that this requires an additional state ϕ, the platform roll angle. Thus the complete vestibular model has the state vector $\boldsymbol{x_{vest}} = [\boldsymbol{x_{scc}}\ \boldsymbol{x_{oto}}\ \phi]^T$, and the state-space model is

$$\boldsymbol{x_{vest}}(k+1) = \boldsymbol{A_{vest}}\boldsymbol{x_{vest}}(k) + \boldsymbol{B_{vest}}\boldsymbol{u_{vest}}(k)$$
$$\boldsymbol{y_{vest}}(k) = \boldsymbol{C_{vest}}\boldsymbol{x_{vest}}(k) \tag{5}$$

with $\boldsymbol{u_{vest}} = \begin{bmatrix} p\ a_y \end{bmatrix}^T$ and $\boldsymbol{y_{vest}} = \begin{bmatrix} \hat{p}\ \hat{a}_y \end{bmatrix}^T$, and state space matrices

$$\boldsymbol{A_{vest}} = \begin{bmatrix} \boldsymbol{A_{scc}} & 0 & 0 \\ 0 & \boldsymbol{A_{oto}} & g\boldsymbol{B_{oto}} \\ 0 & 0 & 1 \end{bmatrix}, \boldsymbol{B_{vest}} = \begin{bmatrix} \boldsymbol{B_{scc}} & 0 \\ 0 & \boldsymbol{B_{oto}} \\ T_S & 0 \end{bmatrix}$$
$$\boldsymbol{C_{vest}} = \begin{bmatrix} \boldsymbol{C_{scc}} & 0 & 0 \\ 0 & \boldsymbol{C_{oto}} & 0 \end{bmatrix} \tag{6}$$

2.1.2 Actuator states

The quantities to be constrained are the actuator lengths and velocities; the platform kinematics must be modelled in order to predict the effect of the control inputs on the

actuators. First, note that the actuator lengths can be directly calculated from the platform position in Cartesian coordinates by vector arithmetic:

$$l_i = \|\boldsymbol{R_{AB}} + \boldsymbol{L}\boldsymbol{B_i} - \boldsymbol{A_i}\|_2 \tag{7}$$

where $\boldsymbol{R_{AB}}$ is the vector from the origin on the ground to the platform centroid, the subscript i indicates actuator number ($i = 1,\ldots,6$), $\boldsymbol{A_i}$ and $\boldsymbol{B_i}$ are the vectors from the origin to the actuator lower joint and platform centroid to the actuator upper joint respectively, and $\boldsymbol{L}$ is the rotation matrix from the ground reference to platform reference; using a small angle approximation, this is

$$\boldsymbol{L} = \begin{bmatrix} 1 & -\psi + \phi\theta & \phi\psi - \theta \\ \psi & 1 & \phi + \theta\psi \\ -\theta & \phi & 1 \end{bmatrix} \tag{8}$$

Combining equations 7 and 8,

$$\begin{aligned} l_i &= \left\| \begin{bmatrix} x_{AB} \\ y_{AB} \\ z_{AB} \end{bmatrix} + \begin{bmatrix} 1 & -\psi + \phi\theta & \phi\psi - \theta \\ \psi & 1 & \phi + \theta\psi \\ -\theta & \phi & 1 \end{bmatrix} \begin{bmatrix} x_{B,i} \\ y_{B,i} \\ z_{B,i} \end{bmatrix} - \begin{bmatrix} x_{A,i} \\ y_{A,i} \\ z_{A,i} \end{bmatrix} \right\|_2 \\ &= \left\| \begin{bmatrix} x_{AB} + x_{B,i} + (-\psi + \phi\theta)y_{B,i} + (\phi\psi - \theta)z_{B,i} - x_{A,i} \\ y_{AB} + \psi x_{B,i} + y_{B,i} + (\phi + \theta\psi)z_{B,i} - y_{A,i} \\ z_{AB} - \theta x_{B,i} + \phi y_{B,i} + z_{B,i} - z_{A,i} \end{bmatrix} \right\|_2 \quad \left(= \left\| \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} \right\|_2 \right) \end{aligned} \tag{9}$$

In order to predict the effect of the candidate control sequence on the actuator positions and velocities over the prediction horizon H_p, an expression is sought that links the actuator velocities to the control input, in this case $[p\ a_y]^T$. The derivative of actuator length can be expressed as

$$\frac{d}{dt}(l_i) = \frac{d}{dt}\left(\sqrt{x_i^2 + y_i^2 + z_i^2}\right) = \frac{\frac{1}{2}\left(\frac{d}{dt}\left(x_i^2 + y_i^2 + z_i^2\right)\right)}{l_i} \tag{10}$$

The numerator of equation (10) is found by differentiating the expressions for x_i,y_i,z_i in equation (9). Noting that $z_{A,i} = z_{B,i} = 0$ for all actuators, this is

$$
\begin{aligned}
\frac{d}{dt}(x_i^2 + y_i^2 + z_i^2) \quad &= 2\{\frac{dx}{dt}(x_{AB} + x_{B,i} + (\phi\theta - \psi)y_{B,i} - x_{A,i}) \\
&+ \frac{dy}{dt}(y_{AB} + \psi x_{B,i} + y_{B,i} - y_{A,i}) \\
&+ \frac{dz}{dt}(z_{AB} - \theta x_{B,i} + \phi y_{B,i}) \\
&+ \frac{d\phi}{dt}(\phi\theta^2 y_{B,i} - \theta\psi y_{B,i} + \theta x_{AB} y_{B,i} + \theta y_{B,i}(x_{B,i} - x_{A,i}) \\
&\qquad + y_{B,i}^2\phi + z_{AB} y_{B,i} - \theta x_{B,i} y_{B,i}) \\
&+ \frac{d\theta}{dt}(\phi^2\theta y_{B,i}^2 - \phi\psi y_{B,i}^2 + \phi x_{AB} y_{B,i} + \phi y_{B,i}(x_{B,i} - x_{A,i}) \\
&\qquad + x_{B,i}^2\theta + z_{AB} x_{B,i} - \phi x_{B,i} y_{B,i}) \\
&+ \frac{d\psi}{dt}(-\phi\theta y_{B,i}^2 + \psi y_{B,i}^2 - x_{AB} y_{B,i} - y_{B,i}(x_{B,i} - x_{A,i}) + x_{B,i}^2\psi \\
&\qquad + y_{AB} x_{B,i} + x_{B,i} y_{B,i} - x_{B,i} y_{A,i})\}
\end{aligned}
\tag{11}
$$

Equation (10) can now be rewritten as

$$
\frac{d}{dt}(l_i) = \frac{k_{A,i}\dot{x} + k_{B,i}\dot{y} + k_{C,i}\dot{z} + k_{D,i}\dot{\phi} + k_{E,i}\dot{\theta} + k_{F,i}\dot{\psi}}{l_i} \tag{12}
$$

As $k_{A\ldots F}$ are functions of the time-varying platform position, orientation and their derivatives, they are themselves time-varying. However, since the velocity variation with position is quite smooth over the relatively short time horizon, the $k_{A\ldots F}$ are evaluated for the current position and assumed constant over the prediction horizon. This linearization means that the problem formulation can continue with a linear state-space model.

One key requirement for the controller is that it must operate in real-time. The computational complexity grows cubically with the number of model states n_x; thus the model is kept as small as possible. To this end, the velocities in the other 4 DOFs are assumed to be zero, such that

$$
\frac{d}{dt}(l_i) = \frac{k_{B,i}\dot{y} + k_{D,i}\dot{\phi}}{l_i} \tag{13}
$$

and the actuator lengths can then be modelled as

$$
\boldsymbol{x_{act,i}}(k+1) = \boldsymbol{x_{act,i}}(k) + T_s\left(\frac{k_{B,i}}{l_i}\dot{y} + \frac{k_{D,i}}{l_i}\dot{\phi}\right) \tag{14}
$$

Thus only seven states must be added to the model; the six actuator lengths and $\dot{y}$ (the roll derivative is already available as one of the inputs to the system). Clearly this simplification affects the constraint handling; multiple controllers may end up applying a combined input that drives one of the actuators beyond its limits. However, initial tests showed that this was not a problem for a range of manoeuvres, with any potential conflict taken care of once the actuator states are updated in the controllers at the next time step. If issues were to arise then a reduction of the specified actuator limits of, say, 5-10% would give an extra safety margin and still allow good workspace usage.

2.1.3 Combined system model

The complete linear system modelling the vestibular dynamics and platform kinematics is

$$\boldsymbol{x_{mpc}}(k+1) = \boldsymbol{A_{mpc}}\boldsymbol{x_{mpc}}(k) + \boldsymbol{B_{mpc}}\boldsymbol{u}(k)$$

$$\boldsymbol{x_{mpc}} = [\boldsymbol{x_{scc}}\ \boldsymbol{x_{oto}}\ \phi\ \dot{y}\ l_1 \dots l_6]^T$$

$$\boldsymbol{A_{mpc}} = \begin{bmatrix} \boldsymbol{A_{vest}} & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & T_s\dfrac{k_B(1)}{l_1} & 1 & 0 & \cdots & 0 \\ 0 & T_s\dfrac{k_B(2)}{l_2} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & T_s\dfrac{k_B(6)}{l_6} & 0 & 0 & \cdots & 1 \end{bmatrix}, \boldsymbol{B_{mpc}} = \begin{bmatrix} \boldsymbol{B_{vest}} & \\ 0 & T_s \\ T_s\dfrac{k_D(1)}{l_1} & 0 \\ T_s\dfrac{k_D(2)}{l_2} & 0 \\ \vdots & \vdots \\ T_s\dfrac{k_D(6)}{l_6} & 0 \end{bmatrix} \quad (15)$$

The derivation for the longitudinal/pitch pair is the same as above; for the vertical and yaw DOFs, the derivation is similar, with the resulting systems given by equations (16) and (17) respectively

$$\boldsymbol{x_{mpc,z}}(k+1) = \boldsymbol{A_{mpc,z}}\boldsymbol{x_{mpc,z}}(k) + \boldsymbol{B_{mpc,z}}u_z(k)$$

$$\boldsymbol{x_{mpc,z}} = [\boldsymbol{x_{oto}}\ \dot{z}\ p_1 \dots p_6]^T$$
$$u_z = a_z$$

$$\boldsymbol{A_{mpc,z}} = \begin{bmatrix} \boldsymbol{A_{oto}} & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & T_s k_C(1) & 1 & 0 & \cdots & 0 \\ 0 & T_s k_C(2) & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & T_s k_C(6) & 0 & 0 & \cdots & 1 \end{bmatrix}, \boldsymbol{B_{mpc,z}} = \begin{bmatrix} \boldsymbol{B_{oto}} \\ T_s \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (16)$$

$$\boldsymbol{x_{mpc,\psi}}(k+1) = \boldsymbol{A_{mpc,\psi}}\boldsymbol{x_{mpc,\psi}}(k) + \boldsymbol{B_{mpc,\psi}}u_\psi(k)$$

$$\boldsymbol{x_{mpc,\psi}} = [\boldsymbol{x_{scc}}\ p_1 \dots p_6]^T$$
$$u_\psi = r$$

$$\boldsymbol{A_{mpc,\psi}} = \begin{bmatrix} \boldsymbol{A_{oto}} & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}, \boldsymbol{B_{mpc,\psi}} = \begin{bmatrix} \boldsymbol{B_{oto}} \\ T_s k_F(1) \\ T_s k_F(2) \\ \vdots \\ T_s k_F(6) \end{bmatrix} \quad (17)$$

The authors note that a ‘complete’ system model should also include the platform dynamics; the input to the vestibular system is the actual motion of the platform, not the demanded motion. However, as noted earlier, it is desirable to keep the state dimension as small as possible in order to reduce the computational burden. Previous tests have shown the platform to have a bandwidth of around 20Hz, so in lieu of a platform dynamics model the time step

for the MPC solver is set to 25ms, i.e. f_s = 40Hz. The resulting smooth band-limited control ensures that the actuator bandwidth limit is respected without adding extra complexity to the MPC problem.

2.2 MPC formulation

The cost function is formulated to apply a cost to motion perception error, platform excursion and control input along the prediction horizon,

$$J = \sum_{i=0}^{H_p} \left\{ \boldsymbol{e}(t+i)^T \boldsymbol{Q_{perc}} \boldsymbol{e}(t+i) + \boldsymbol{x_{plat}^T}(t+i) \boldsymbol{Q_{plat}} \boldsymbol{x_{plat}}(t+i) + \boldsymbol{u}(t+i)^T \boldsymbol{R} \boldsymbol{u}(t+i) \right\} \tag{18}$$

where $\boldsymbol{e}(t)=\boldsymbol{y_{vest,car}}(t) - \boldsymbol{y_{vest,plat}}(t)$ is the perception error, and $\boldsymbol{Q_{perc}}$, $\boldsymbol{Q_{plat}}$ and $\boldsymbol{R}$ are weighting matrices. Noting from equations (3), (4) and (6) that the perceived roll velocity and lateral acceleration are simply elements of the $\boldsymbol{x_{mpc}}$ state vector, the cost function can be rewritten as

$$J = \sum_{i=0}^{H_p} \|\boldsymbol{y}(t+i)\|_{\boldsymbol{Q}}^2 + \|\boldsymbol{u}(t+i)\|_{\boldsymbol{R}}^2 \tag{19}$$

where $\boldsymbol{y}(t) = \boldsymbol{x_{mpc}}(t) - \boldsymbol{x_{ref}}(t)$, the difference between the state vector and the desired reference. The reference is set to $\boldsymbol{x_{ref}}(t) = [\hat{p}_{ref}\ 0\ 0\ \hat{a}_{y,ref}\ 0\ 0\ 0\ \boldsymbol{x_{act,0}}]$, $\boldsymbol{x_{act,0}}$ being the vector of actuator neutral lengths; the reference is assumed constant over the whole prediction horizon. The $\boldsymbol{Q}$ and $\mathbf{R}$ matrices are diagonal; to simplify the tuning process, they are written as

$$\begin{aligned} \boldsymbol{Q} &= diag\left(\left[Q_{perc,p}\ 0\ 0\ Q_{perc,y}\ 0\ 0\ 0\ K_{plat}\boldsymbol{Q_{plat}}\right]\right) \\ \boldsymbol{R} &= diag\left(K_{input}\left[R_p\ R_{a_y}\right]\right) \end{aligned} \tag{20}$$

where $\boldsymbol{Q_{plat}}$ is a vector with six equal elements (the cost weights applied to the six actuator terms) and all other values are scalar. Nominal values for the individual Q and R terms (listed in Appendix B) were chosen during initial testing, and then fixed; subsequent fine tuning is then done using only the scale factors K_{plat} and K_{input}, i.e. the platform and input costs are tuned relative to the perception error cost.

The MPC problem can now be stated as

$$\min_{\boldsymbol{u}(t)} \ J(\boldsymbol{y}(t), \boldsymbol{u}(t))$$

$$\begin{aligned} subject\ to \quad \boldsymbol{x_{mpc}}(t+i+1) &= \boldsymbol{A_{mpc}}\boldsymbol{x_{mpc}}(t+i) + \boldsymbol{B_{mpc}}\boldsymbol{u}(t+i) && i = 0, \dots, H_p \\ \boldsymbol{y}(t+i) &= \boldsymbol{x_{mpc}}(t+i) - \boldsymbol{x_{ref}}(t+i) && i = 0, \dots, H_p \\ \boldsymbol{x_{ref}}(t+i) &= \boldsymbol{x_{ref}}(t) && i = 0, \dots, H_p \\ \boldsymbol{u_{min}} \le \boldsymbol{u}(t+i) &\le \boldsymbol{u_{max}} && i = 0, \dots, H_u - 1 \\ \boldsymbol{u}(t+i) &= 0 && i = H_u, \dots, H_p \\ \boldsymbol{x_{act,min}} \le \boldsymbol{x_{act}}(t+i) &\le \boldsymbol{x_{act,max}} && i = 0, \dots, H_p \\ \boldsymbol{v_{act,min}} \le \boldsymbol{v_{act}}(t+i) &\le \boldsymbol{v_{act,max}} && i = 0, \dots, H_p \end{aligned} \tag{21}$$

In order to solve equation (21) the MPC is reformulated as a QP problem. Following the method of Wang and Boyd [11], an optimization variable $\boldsymbol{z}$ is defined as

$$\boldsymbol{z} = \left[\boldsymbol{u}(t), \dots, \boldsymbol{u}(t + H_u - 1), \boldsymbol{y}(t + 1), \dots, \boldsymbol{y}\left(t + H_p\right)\right]^T \tag{22}$$

The QP can then be expressed as

$$\begin{aligned} &\min_{\boldsymbol{z}} \boldsymbol{z}^T \boldsymbol{H} \boldsymbol{z} \\ &subject\ to \quad \boldsymbol{A}\boldsymbol{z} \leq \boldsymbol{b} \\ &\qquad\qquad \boldsymbol{A_{eq}}\boldsymbol{z} = \boldsymbol{b_{eq}} \end{aligned} \tag{23}$$

where, noting that $\mathbf{1}_n$ is a column vector of ones with n elements, $\boldsymbol{I}_{mxn}$ is an m x n identity matrix, and $\otimes$ is the Kronecker product,

$$\mathbf{H} = \begin{bmatrix} \boldsymbol{R} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{R} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{R} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{Q} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{Q} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{Q} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{Q_f} \end{bmatrix}$$

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{I_U} & \mathbf{0} \\ -\boldsymbol{I_U} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{I_A} \\ \mathbf{0} & -\boldsymbol{I_A} \\ \boldsymbol{A_\phi} & \boldsymbol{A_y} \\ -\boldsymbol{A_\phi} & -\boldsymbol{A_y} \end{bmatrix}, \text{ where } \quad \boldsymbol{I_U} = \boldsymbol{I}_{H_u n_{in} \times H_u n_{in}}, \ \boldsymbol{I_A} = \left\{\boldsymbol{I}_{H_p \times H_p} \otimes [\mathbf{0}_{p_c \times n_x - p_c} \quad \boldsymbol{I}_{p_c \times p_c}]\right\}$$

$$\boldsymbol{A_\phi} = \left[\begin{matrix} \left\{I_{H_u \times H_u} \otimes [k_D \ 0]\right\} \\ 0 \end{matrix}\right], \ \boldsymbol{A_y} = \left\{\boldsymbol{I}_{H_p \times H_p} \otimes [0 \ k_D \ 0]\right\}$$

$$\boldsymbol{b} = \begin{bmatrix} \boldsymbol{U_{max}} \\ -\boldsymbol{U_{min}} \\ \boldsymbol{X_{max}} \\ -\boldsymbol{X_{min}} \\ \boldsymbol{V_{max}} \\ -\boldsymbol{V_{min}} \end{bmatrix}, \text{ where } \boldsymbol{U_{max}} = \mathbf{1}_{n_{in} H_u} \otimes \begin{bmatrix} p_{max} \\ a_{y,max} \end{bmatrix}, \boldsymbol{U_{min}} = -\boldsymbol{U_{max}},$$

$\boldsymbol{X_{max}} = \mathbf{1}_{H_p} \otimes \{l_{max} \mathbf{1}_6\}$, $\boldsymbol{X_{min}} = -\boldsymbol{X_{max}}$, $\boldsymbol{V_{max}} = \mathbf{1}_{H_p} \otimes \{\dot{l}_{max} \mathbf{1}_6\}$ and $\boldsymbol{V_{min}} = -\boldsymbol{V_{max}}$

$$\boldsymbol{A_{eq}}$$

$$= \begin{bmatrix} -\boldsymbol{B_{mpc}} & \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & -\boldsymbol{B_{mpc}} & \cdots & \boldsymbol{0} & -\boldsymbol{A_{mpc}} & \boldsymbol{I} & \cdots & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & -\boldsymbol{B_{mpc}} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & -\boldsymbol{A_{mpc}} & \boldsymbol{I} & \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & -\boldsymbol{A_{mpc}} & \boldsymbol{I} & \cdots & \boldsymbol{0} & \boldsymbol{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \cdots & -\boldsymbol{A_{mpc}} & \boldsymbol{I} \end{bmatrix}$$

$$\boldsymbol{b_{eq}} = \begin{bmatrix} \boldsymbol{A_{mpc}}\boldsymbol{x_{mpc}}(t) - \boldsymbol{x_{ref}}(t) \\ \left(\boldsymbol{A_{mpc}} - \boldsymbol{I}\right)\boldsymbol{x_{ref}}(t+1) \\ \vdots \\ \left(\boldsymbol{A_{mpc}} - \boldsymbol{I}\right)\boldsymbol{x_{ref}}(t + H_U - 2) \\ \left(\boldsymbol{A_{mpc}} - \boldsymbol{I}\right)\boldsymbol{x_{ref}}(t + H_U - 1) \\ \vdots \\ \boldsymbol{x_{ref}}(t + H_P - 1) \end{bmatrix}$$

As mentioned earlier the reference is assumed constant over the prediction horizon, so the expression for $\boldsymbol{b_{eq}}$ simplifies to

$$\boldsymbol{b_{eq}} = \begin{bmatrix} \boldsymbol{A_{mpc}}\boldsymbol{x_{mpc}}(t) - \boldsymbol{x_{ref}}(t) \\ \left(\boldsymbol{A_{mpc}} - \boldsymbol{I}\right)\boldsymbol{x_{ref}}(t) \\ \vdots \\ \left(\boldsymbol{A_{mpc}} - \boldsymbol{I}\right)\boldsymbol{x_{ref}}(t) \\ \left(\boldsymbol{A_{mpc}} - \boldsymbol{I}\right)\boldsymbol{x_{ref}}(t) \\ \vdots \\ \boldsymbol{x_{ref}}(t) \end{bmatrix} \tag{24}$$

2.3 QP solver

The solver chosen is a primal barrier, infeasible start Newton method as described by Boyd and Vandenberghe [12] and applied to MPC by Wang and Boyd [11]; the choice is based on the requirement for a solver that can run in real-time on the simulator hardware and be compatible with MATLAB Real-Time Workshop (RTW). Note however that other QP solvers could be used for this problem depending on the requirements of the particular application.

In the primal barrier method, the inequality constraint $\boldsymbol{Az} \leq \boldsymbol{b}$ is replaced by a barrier function; the problem becomes

$$\begin{aligned} &\min_{\boldsymbol{z}} \boldsymbol{z}^T\boldsymbol{H}\boldsymbol{z} + \kappa\Lambda(\boldsymbol{z}) \\ &subject\ to \quad \boldsymbol{A_{eq}}\boldsymbol{z} = \boldsymbol{b_{eq}} \end{aligned} \tag{25}$$

where $\kappa > 0$ is the barrier coefficient and $\Lambda(\mathbf{z})$ is the logarithmic barrier function

$$\Lambda(\boldsymbol{z}) = \sum_i -\log(\boldsymbol{b_i} - \boldsymbol{a_i z}) \tag{26}$$

where the $\boldsymbol{a_i}$,$\boldsymbol{b_i}$ are the rows of the $\boldsymbol{A}$ and $\boldsymbol{b}$ matrices (thus when any of the inequalities is not satisfied, $\Lambda(\mathbf{z}) = \infty$). The basic primal barrier method solves equation (25) for decreasing values of κ until the desired accuracy is achieved (as $\kappa \to 0$, the solution converges to the optimum feasible point); Wang and Boyd [11] show that, in combination with techniques such as warm starting, solver speed can be increased with little loss in control quality by fixing the κ value. Here, in order to guarantee robustness, a compromise is used; the solver begins with a large κ value ($\kappa=10^3$), then once the residual norm is less than a chosen threshold value, κ is reduced by a factor of 10 and the optimization continues. This process is repeated until an iteration limit is reached. This method is robust for large inputs, with several iterations at high κ values resulting in a sub-optimal but feasible solution, whereas a small input tends to reach smaller κ values and hence a more optimal solution.

The equality-constrained QP of equation (25) is solved using an infeasible start Newton method, so-called because the starting point is chosen to satisfy the inequality constraints but not necessarily the equality constraints. Considering a dual variable v for the equality constraint, the Lagrangian for the QP of equation (25) is

$$L(\boldsymbol{z}, \boldsymbol{\nu}) = z^T H z + \kappa\Lambda(\boldsymbol{z}) + \boldsymbol{\nu}^T(\boldsymbol{A_{eq}z} - \boldsymbol{b_{eq}}) \tag{27}$$

For a solution $\boldsymbol{z}^*$,$\boldsymbol{\nu}^*$ to be optimal, $\nabla_z L(\boldsymbol{z}^*, \boldsymbol{\nu}^*)$ must be zero; using the notation of primal and dual residuals $\boldsymbol{r_p}$ and $\boldsymbol{r_d}$, the optimal solution is therefore found when

$$\boldsymbol{r_p} = \boldsymbol{A_{eq}z^*} - \boldsymbol{b_{eq}} = \mathbf{0} \qquad \boldsymbol{r_d} = 2\boldsymbol{Hz^*} + \kappa\boldsymbol{A^T d}(\boldsymbol{z^*}) + \boldsymbol{A_{eq}^T \nu^*} = \mathbf{0} \tag{28}$$

where $\boldsymbol{A}$ is as defined in equation (23) and $d_i(\boldsymbol{z}) = 1/(\boldsymbol{b_i}\text{-}\boldsymbol{a_i z})$. An approximation to $\boldsymbol{z}^*$ is found using a fixed number of Newton's method iterations, with κ varied as described above. The Newton step $\boldsymbol{x_{NS}} = [\Delta\boldsymbol{z}\ \Delta\boldsymbol{\nu}]^T$ is found by setting the Taylor series approximation of the residual to zero, i.e. $\boldsymbol{r}(\boldsymbol{z}, \boldsymbol{\nu}) + \nabla\boldsymbol{r}(\boldsymbol{z}, \boldsymbol{\nu})\boldsymbol{x_{NS}} = \mathbf{0}$ where $\boldsymbol{r}(\boldsymbol{z}, \boldsymbol{\nu}) = [\boldsymbol{r_d} \quad \boldsymbol{r_p}]^T$. Expanding this expression gives the equation to be solved at each iteration:

$$\begin{bmatrix} 2\boldsymbol{H} + \kappa\boldsymbol{A^T} diag(\boldsymbol{d})^2\boldsymbol{A} & \boldsymbol{A_{eq}^T} \\ \boldsymbol{A_{eq}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{z} \\ \Delta\boldsymbol{\nu} \end{bmatrix} = -\begin{bmatrix} \boldsymbol{r_d} \\ \boldsymbol{r_p} \end{bmatrix} \tag{29}$$

The solution is found using block elimination; defining $\boldsymbol{\Phi} = 2\boldsymbol{H} + \kappa\boldsymbol{A^T} diag(\boldsymbol{d})^2\boldsymbol{A}$, the Schur complement $\boldsymbol{Y} = \boldsymbol{A_{eq}}\boldsymbol{\Phi^{-1}}\boldsymbol{A_{eq}^T}$ and $\boldsymbol{\beta} = -\boldsymbol{r_p} + \boldsymbol{A_{eq}}\boldsymbol{\Phi^{-1}}\boldsymbol{r_d}$ is formed. The Newton step is then found from $\boldsymbol{Y}\Delta\boldsymbol{\nu} = -\boldsymbol{\beta}$ and $\boldsymbol{\Phi}\Delta\boldsymbol{z} = -\boldsymbol{r_d} - \boldsymbol{A_{eq}^T}\boldsymbol{\nu}$. Computationally this is reasonably fast, as $\boldsymbol{\Phi}$ is block diagonal and $\boldsymbol{Y}$ is block tridiagonal, hence the inverses are more easily found (note that the zeros in the main diagonal of the $\boldsymbol{Q}$ matrix, equation (20), must be given some small value, in this case 10^{-9}, in order to avoid a singular $\boldsymbol{\Phi}$ matrix).

A line search method finds the step size s, $0 < s \leq 1$, to take in the descent direction (i.e. $\boldsymbol{z_{n+1}} = \boldsymbol{z_n} + s\Delta\boldsymbol{z}$, $\boldsymbol{\nu_{n+1}} = \boldsymbol{\nu_n} + s\Delta\boldsymbol{\nu}$); an inexact but fast method, the backtracking line search from Boyd and Vandenberghe [12], is used. Beginning with $s = 1$, the line search reduces the step length by the factor β until the condition

$$\|\boldsymbol{r}(\boldsymbol{z}+s\Delta\boldsymbol{z},\boldsymbol{v}+s\Delta\boldsymbol{v})\|_2 < (1-\alpha s)\|\boldsymbol{r}(\boldsymbol{z},\boldsymbol{v})\|_2 \tag{30}$$

is met (where $0<\alpha<0.5$ and $0<\beta<1$).

3 Quantitative testing

3.1 Algorithm characterisation

The performance of the QP solver of section 2.3 is compared with that of the MATLAB Optimization Toolbox QP solver 'quadprog'. Vehicle lateral acceleration data taken from an earlier simulator test was used as the reference for the MPC lateral/roll controller, and control sequences were generated using three methods of solving the QP: the solver described above with a high iteration limit; the same solver with an iteration limit that was found to allow real-time running on the simulator; and MATLAB 'quadprog' running an interior point algorithm. Figure 3 shows the controls generated by the three different solvers.

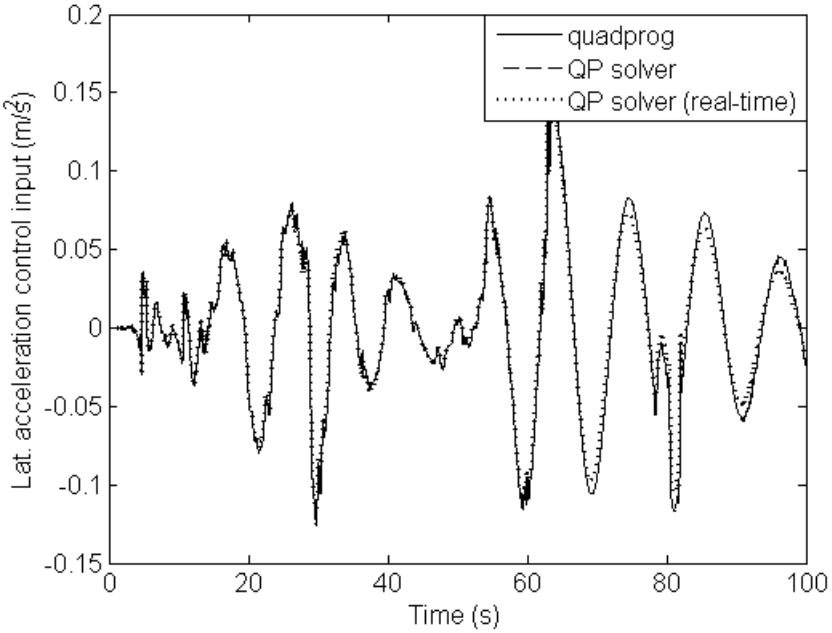

Figure 3 – Comparison of controls generated by new QP solver and MATLAB 'quadprog'

With a large iteration limit, the solver implemented above gives the same solution as 'quadprog' (this is to be expected as the QP is convex and thus has a unique solution). When the iteration limit is reduced to allow for real-time operation, the accuracy of the solution is only reduced by a small amount. The small degree of sub-optimality observed here is judged as being acceptable; Wang and Boyd (2010) note that MPC is an heuristic for finding a good control input, and in practice a good approximation to the solution to the QP is sufficient.

The robustness of the MPC algorithm to large vehicle motions is tested by applying 15s acceleration pulses of amplitude $1 m/s^2$ and $100 m/s^2$; the actuator lengths for this test are shown in figure 4. Note that even in the latter extreme case the platform does not exceed the actuator length limit.

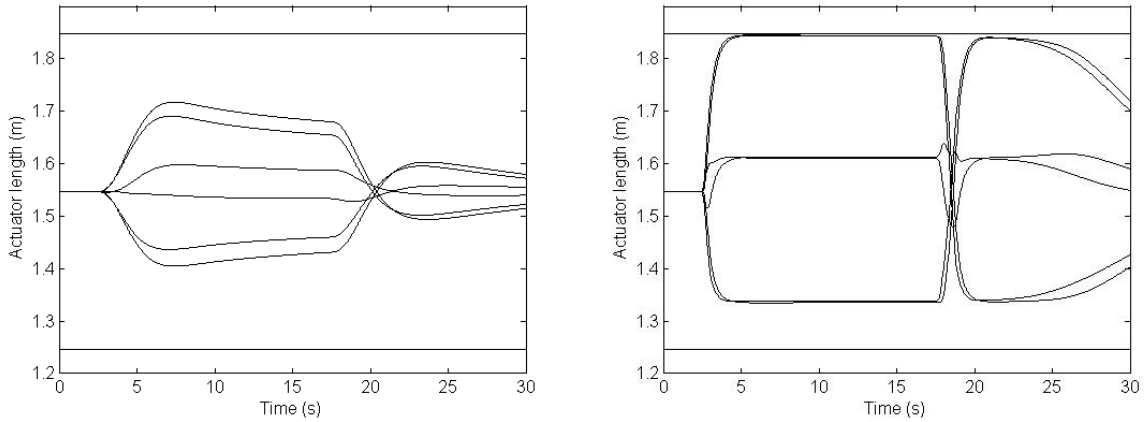

Figure 4 – Actuator lengths for 15s acceleration pulse, amplitude 1m/s^2 and 100m/s^2; horizontal lines indicate actuator limits

3.2 Offline comparison with classical algorithm

The response of the MPC algorithm is now compared with that of the classical algorithm. The classical algorithm was tuned for a worst-case 10m/s^2 acceleration step; the cut-off frequency and damping parameters were chosen to match those of the simulator manufacturer's classical-based cueing algorithm and the gain selected such that the platform position and velocity limits were respected for the worst case.

First, a 1m/s^2 15s acceleration pulse is applied to both algorithms as the lateral acceleration reference. Figure 5 shows the perceived motion and figure 6 the platform excursion. The classical algorithm is tuned to give only the high-frequency transients, as reflected in the perceived motion plot. The MPC algorithm on the other hand achieves a good match to the shape of the motion perception in the real vehicle.

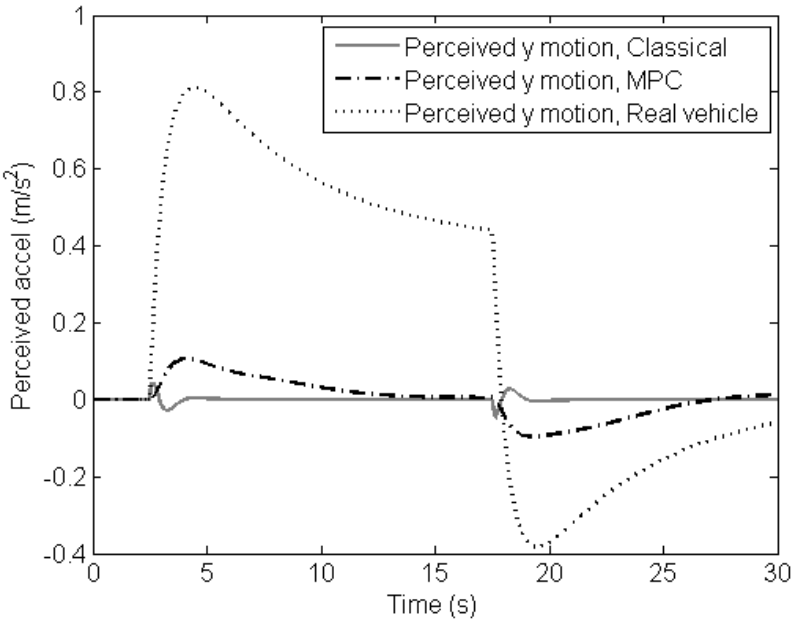

Figure 5 – Perceived motion for 1m/s^2 acceleration pulse

Due to the worst-case being much higher than the actual input, the classical algorithm makes poor use of the platform workspace. Of course the classical algorithm could have been tuned for the 1m/s^2 pulse; the point made here is that the classical algorithm must be tuned for a worst case, whereas the MPC algorithm requires no tuning for different driving scenarios. In reality the worst-case anticipated vehicle motion is often of much greater amplitude than, say, the RMS acceleration, and so for most simulations the classical workspace usage is poor. Conversely, the MPC algorithm can deal with an input acceleration of any magnitude and still respect the actuator limits while making better use of the workspace at lower input accelerations. The MPC tuning gains K_{plat} and K_{input} are used to tune how aggressively the algorithm exploits the workspace (i.e. is it conservative or does it tend to use most of the workspace even for low accelerations); tuning these parameters is more intuitive than tuning the classical filter parameters, as their effect on the algorithm behaviour is more obvious.

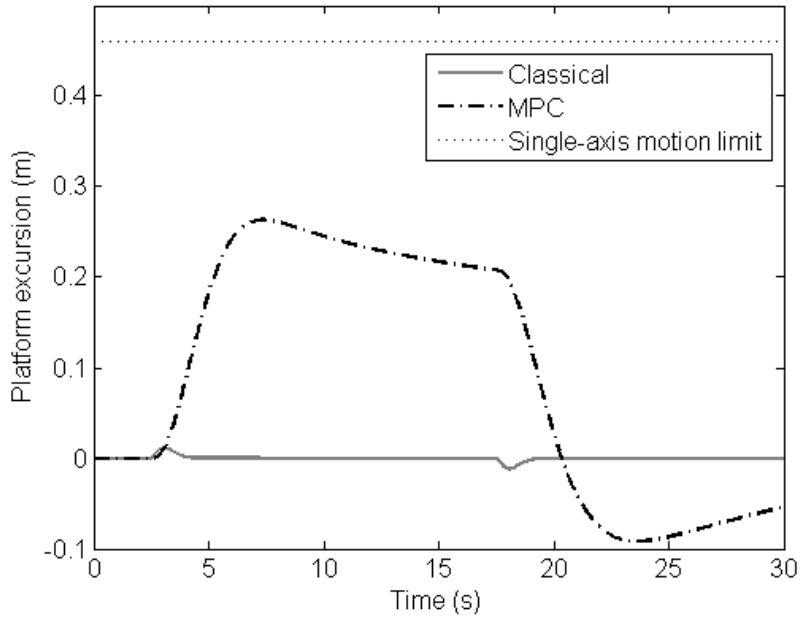

Figure 6 – Platform excursion for 1m/s^2 acceleration pulse

Now consider lateral acceleration data from a simulated lap around a race circuit, where the maximum acceleration is around 1g. Figure 7 shows the motion perception for both algorithms compared with that in the real vehicle. As with the pulse response above, the classical algorithm produces a control sequence with more high frequency content than MPC, but the MPC algorithm matches the perceived acceleration profile more closely than the classical algorithm.

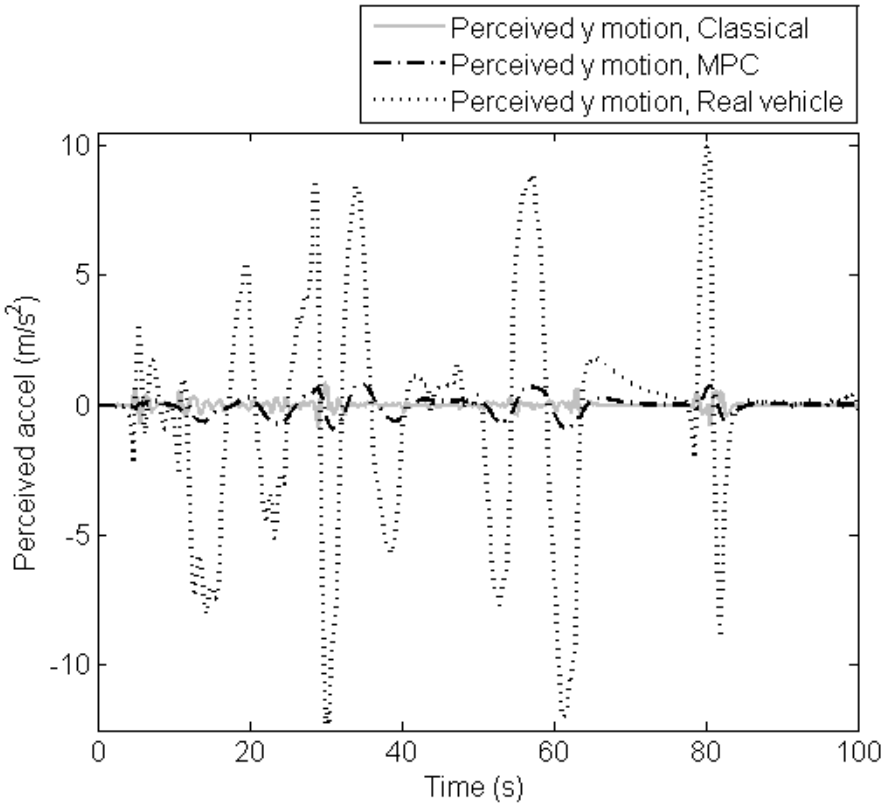

Figure 7 – Perceived lateral acceleration for circuit data

3.3 Comparison with body sideslip algorithm

In section 4 we will consider qualitative performance of the MPC and classical algorithms compared with the body sideslip (BSS) algorithm – an algorithm developed previously by the authors which, unlike most motion cueing algorithms, is not based around acceleration tracking; instead the focus is on providing good information about the vehicle state to the driver. The vehicle sideslip angle is a key quantity in the evaluation of vehicle stability, and so this angle is used directly as the platform yaw demand – since this angle fits comfortably within the platform yaw limit for all but the most extreme cases, there is no need to modify or filter the signal.

In order to provide the necessary position, velocity and acceleration signals, the first and second derivatives of sideslip angle are required. Using the small angle approximation $\beta = v/u$, these are

$$\dot{\boldsymbol{\beta}} = \frac{\dot{\boldsymbol{v}}}{\boldsymbol{u}} - \frac{\dot{\boldsymbol{u}}\boldsymbol{v}}{\boldsymbol{u^2}} \qquad\qquad \ddot{\beta} = \frac{\ddot{v}}{u} - \frac{\ddot{u}v}{u^2} - 2\frac{\dot{u}\dot{v}}{u^2} + 2\frac{\dot{u}^2 v}{u^3} \tag{31}$$

At first the second derivative appears problematic as there are jerk terms $\ddot{u}$ and $\ddot{v}$. However, taking the derivative of the expression for lateral acceleration a_y, it is seen that

$$a_y = \frac{\sum F_y}{m} = \dot{v} + ur + wp$$
$$\xrightarrow{d/dt} \quad \ddot{v} = \frac{\sum \dot{F}_y}{m} - \dot{u}r - u\dot{r} - \dot{w}p - w\dot{p} \tag{32}$$

All terms in this expression are available in the vehicle model, including the tyre force derivatives which are available due to the inclusion of tyre relaxation lag dynamics

$$\hat{\dot{F}}_{xi,yi} = \tau^{-1}\left(F_{xi,yi} - \hat{F}_{xi,yi}\right) \tag{33}$$

with the derivation for $\ddot{u}$ following the same procedure with a_x.

Clearly the body sideslip yaw cue does not provide a complete 6-DOF cue; it is necessary to add motion in the remaining DOF by other means. The algorithm as used in these tests uses vehicle roll angle as the platform roll demand, again with no filtering; longitudinal, pitch and vertical motion is provided by the classical algorithm.

To provide some comparison of the BSS algorithm response, the perceived yaw rate for the three algorithms for an ISO double lane change manoeuvre at 20 m/s is shown in figure 8. The vehicle parameters are for a mid-sized front wheel drive car and are based on manufacturer specifications and vehicle test data.

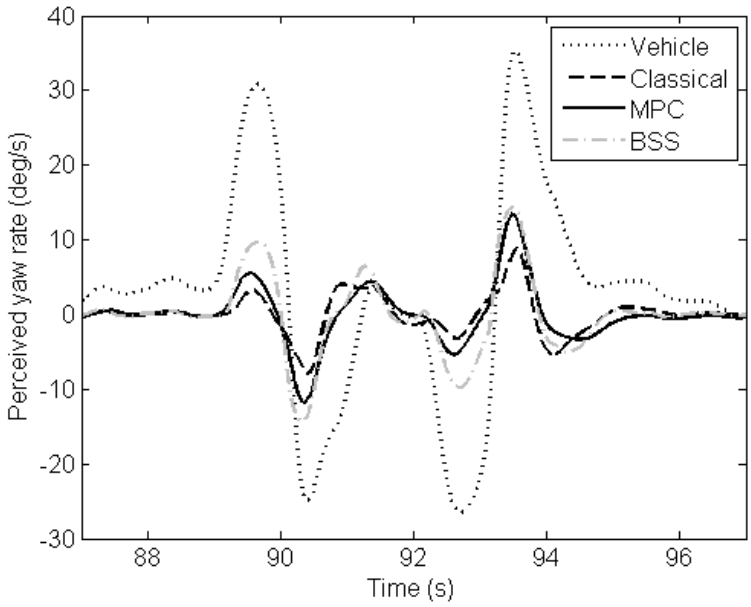

Figure 8 – Double lane change perceived yaw rate

Although the BSS algorithm appears to match perceived yaw rate reasonably well, this is not by design, and this plot is not intended to compare perception matching performance; the significant difference in the algorithm concepts means that there is no fully objective comparison, and this result is merely presented as an indication of how the algorithm responses compare.

4 Qualitative performance

The MPC algorithm is now evaluated in driver-in-the-loop tests; the comparator algorithms are the simulator manufacturer's implementation of the classical algorithm with adaptive gains, and the body sideslip algorithm introduced in section 3.3.

4.1 Simulator details

The Loughborough simulator is based on an electrically-driven Stewart platform; additional sensory feedback is provided by three computer screens, a steering wheel with torque feedback, and a three-speaker audio system. The simulation runs on three desktop PCs, one of which is the main simulation computer and also generates graphics for the centre screen, with the other two running as slaves to generate the graphics for the two outer screens. Additionally, two real-time computers control the motion platform and steering feedback motor.

Motion in the six DOF are dealt with separately by the motion controller; if the manufacturer cueing is to be used, the vehicle acceleration in that DOF is sent to the platform controller. If another (user-defined) cueing algorithm is to be used, a position, velocity and acceleration demand for that DOF is sent to the controller instead. It is therefore possible to use a mixture of manufacturer and user-defined cueing across the six DOF. The vehicle dynamics and custom motion cueing are both defined in a Simulink model, which is compiled with Real-Time Workshop to create a plug-in for the manufacturer-supplied simulation software. An additional software interface allows the operator to send signals into the Simulink model during runtime, which allows the choice of cueing algorithm to be changed during a simulation.

The vehicle model used in the tests has a body free to move in 6 DOF; each of the four wheels has a spin DOF (two of which, either the front or rear pair, have the drivetrain output torque acting on them) and a vertical DOF. The vertical loads are calculated based on a simple spring-damper suspension model, the vertical load along with lateral and longitudinal slips being used to calculate tyre forces using a combined slip Pacejka model, full details of which are in [13]. Tyre parameters are estimated based on vehicle test data; a few different sets of vehicle parameters are used in the tests below and are noted in the appropriate section.

4.2 Driver-in-the-loop testing

In order to evaluate the algorithms over a range of conditions, two separate tests were conducted, one for driving in the linear region of vehicle behaviour and the other in the non-linear region. The focus of the tests is on cornering motion, and as such the manufacturer cueing is used for longitudinal, pitch and vertical motion in all cases, with lateral, roll and yaw motion provided by the cueing algorithm under test.

The tests are carried out as bidirectional pairwise comparisons; each test block consists of seven laps, within which each of the three possible pairings of the three algorithms is tested in both directions. The lap order is varied for each run to eliminate order effects. The tests are carried out blind, i.e. the operator is aware which algorithm is in use but the participant is not.

After each lap, the participants were asked to rate how realistic the simulator felt on that lap compared to the previous lap using a Likert scale, figure 9.

How realistic was the simulator compared to the previous lap?				
Much worse	Worse	About the same	Better	Much better
-2	-1	0	+1	+2

Figure 9 – Likert scale for simulator experiments

4.2.1 Linear testing

Seven of the ten most consistent participants from a previous set of tests were selected to take part. The participants were asked to drive as they would on the public road; the course chosen is a 1.6 km section of mountain road with a series of tight corners. The vehicle parameters are matched approximately to a large saloon car. Each participant was given five laps with the classical cueing to get used to the vehicle behaviour and track layout, and then each completed a block of seven pairwise test laps. The ratings for each participant are normalized by the mean absolute value of their non-zero responses to eliminate differing interpretations of the Likert range. For each pair test, the winning algorithm's total score is increased by the normalized score and the losing algorithm's score decreased by the same amount. The resulting total scores are shown in figure 10.

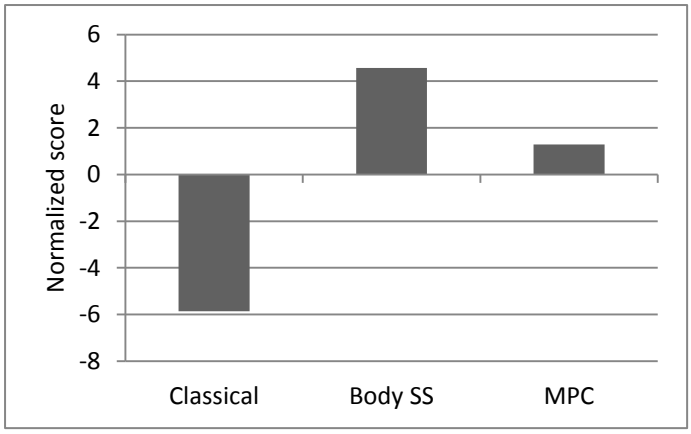

Figure 10 – Total scores for linear testing

Both the MPC and BSS algorithms appear to perform well relative to the classical algorithm. Calculation of the Friedman statistic (a non-parametric form of analysis of variance, see e.g. [14]) for these results gives a p-value of 0.746. This suggests that the null hypothesis (that there is no difference in the algorithm rating) cannot be rejected with certainty, and the result is therefore treated with a degree of caution. This is in agreement with previous tests where the BSS algorithm is compared with several established algorithms in the linear handling regime [15], where no strong preference for any of the algorithms was observed for this type of driving.

4.2.2 Non-linear testing

For the non-linear tests, a group of 14 participants was used, comprising approximately equal numbers of ‘normal’, ‘advanced’ and ‘expert’ drivers; respectively these are drivers with road driving experience only (and no/very little experience of non-linear vehicle behaviour), some experience of non-linear behaviour, and extensive experience of sustained driving near the limit of vehicle capability. Two vehicle parameter sets were selected, the front-wheel-drive saloon car used in section 3.3, and a rear-wheel-drive single-seat race car with a representative set of vehicle parameters based on data from the British Formula 3 series [16]. A 2.1 km race circuit was chosen as the course, with the intention that this environment would encourage drivers to operate the vehicle in the non-linear region.

This time, the participants drove three laps with the classical cueing to gain some experience of the track and car, then a block of seven pairwise testing laps to evaluate the algorithms, followed by three more laps with the classical cueing and a second set of seven pairwise laps. This procedure was repeated for both vehicles. The total normalized scores for the three groups of participants, separated by vehicle type, are shown in figure 11.

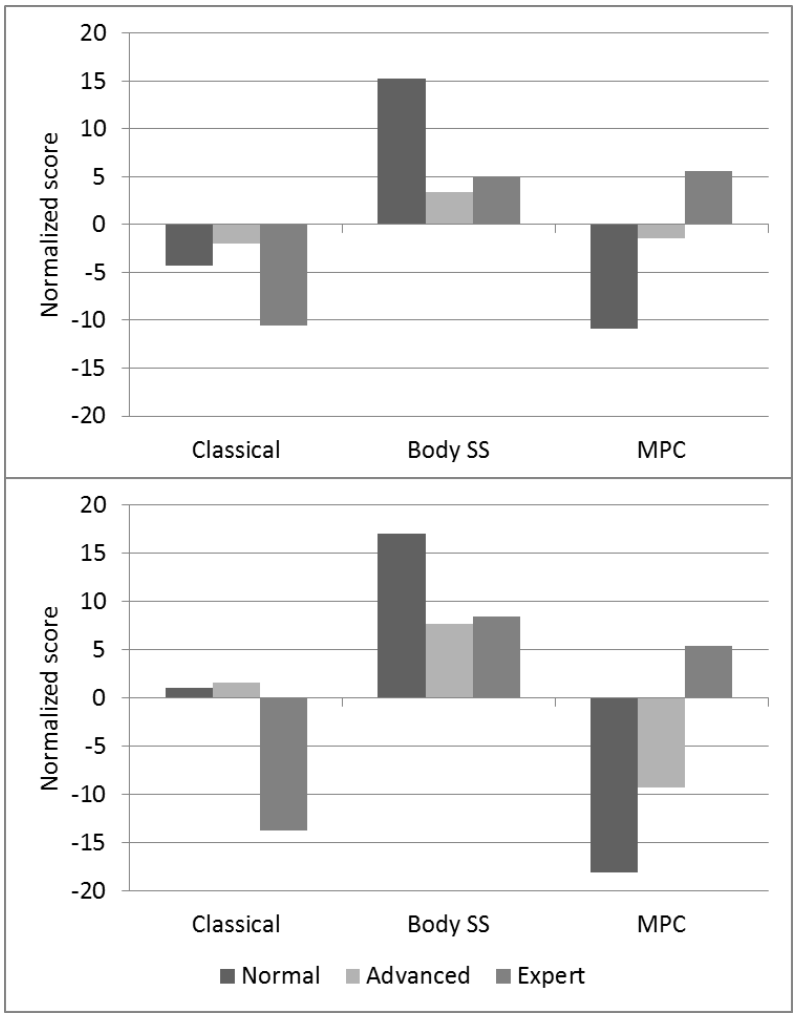

Figure 11 – Total normalized scores separated by participant group (top: saloon car, bottom: race car)

Overall, there is an apparent preference for the feedback provided by the BSS algorithm. An interesting pattern is observed in the MPC results; it would appear that the more experienced the driver group, the higher the preference for MPC, with the expert group having similar positive scores for MPC and BSS and a large negative result for the classical algorithm. This is a promising result as the expert drivers are much better placed to rate the realism of the simulator for this particular test case.

The Friedman test for these data yields a p-value of 0.0008 or 0.0014, depending on whether the results are ordered with each of the four runs or each of the participants treated as a 'block'. These are well below the suggested threshold of 0.05 (see e.g. [17]) and indicate that the null hypothesis can be rejected with some confidence here.

It is worth repeating here the point made in section 3.3 that the BSS algorithm only provides a cue in the yaw DOF, and so although this extra yaw cue clearly has some value, this is in addition to the motion in the other DOF that must be provided by another algorithm. Based on these results, a good choice of algorithm would include the BSS algorithm in the yaw DOF and, in the remaining DOF, MPC would appear to be a better choice than the classical algorithm. Future work should aim to evaluate this possibility, as well as investigate the suitability of MPC in the longitudinal, pitch and vertical DOF (a comparison of full six-DOF MPC vs. six-DOF classical motion would be of use).

5 Conclusions

A motion cueing algorithm based on a Model Predictive Control formulation has been derived, where the motion perceived in the simulator tracks that perceived in the real vehicle whilst respecting the platform workspace limits; additionally, a solver for the MPC problem was implemented that reduces the problem to iterations of Newton's method. Of note is the algorithm's use of actuator positions and velocities, rather than Cartesian states, as constraints; the result is a controller that makes good use of the available workspace. The algorithm is also easy to tune; once the actuator limits have been set and preliminary tuning carried out, only two cost parameters, K_{plat} and K_{input}, must be tuned. The effect of these parameters is also more obvious than, say, the filter parameters of the classical algorithm.

The MPC algorithm is compared with the classical algorithm and the BSS algorithm developed previously by the authors. Simulation tests reveal that the motion perception tracking works well, the MPC algorithm providing a much closer match to the perceived vehicle motion than the classical algorithm. In driver-in-the-loop tests, the MPC algorithm is rated better than the classical algorithm for driving in the linear handling regime, a result also seen for a group of experienced drivers for tests in the non-linear handling regime. The BSS algorithm comes out best, but it is noted that this is a single-DOF cue that must be supplemented by other types of motion cueing in the remaining DOF.

An extension of this work should extend the comparison with the classical algorithm to all six DOF, and investigate the suitability of a cueing algorithm based around MPC with the addition of the BSS yaw cue.

References

[1]	Newton, A. P., & Best, M. C. (2006). *The influence of motion on handling dynamics analysis in full vehicle simulators.* Paper presented at the 8th International Symposium on Advanced Vehicle Control (AVEC), Taipei, Taiwan, http://hdl.handle.net/2134/8329
[2]	Siegler, I., Reymond, G., Kemeny, A. & Berthoz, A. (2001). *Sensorimotor integration in a driving simulator: contributions of motion cueing in elementary driving tasks.* Paper presented at the Driving Simulation Conference DSC2001, Nice, France.
[3]	Schmidt, S. F., & Conrad, B. (1970). *Motion Drive Signals for Piloted Flight Simulators.* (NASA Report. CR-1601).
[4]	Parrish, R. V., Dieudonne, J. E., Bowles, R. L. and Martin, D. J. (1975). *Coordinated Adaptive Washout for Motion Simulators.* Journal of Aircraft, 12(1), 44-50. doi:10.2514/3.59800
[5]	Sivan, R., Ish-Shalom, J., & Huang, J. K. (1982). *An Optimal Control Approach to the Design of Moving Flight Simulators.* IEEE Transactions on Man, Systems and Cybernetics, 12(6), 818-827. doi:10.1109/TSMC.1982.4308915
[6]	Grant, P., & Naseri, A. (2005). *Actuator State Based Adaptive Motion Drive Algorithm.* Paper presented at the Driving Simulation Conference DSC2005, Florida.
[7]	Dagdelen, M., Reymond, G., Kemeny, A., Bordier, M., Maïzi, N. (2009). *Model-based predictive motion cueing strategy for vehicle driving simulators*. Control Engineering Practice, 17(9), 995-1003. doi:10.1016/j.conengprac.2009.03.002
[8]	Gutman, P. O. & Cwikel, M. (1987). *An algorithm to find maximal state constraint sets for discrete-time linear dynamical systems with bounded controls and states*. IEEE Transactions on Automatic Control, 32(3), 251-254. doi:10.1109/TAC.1987.1104567
[9]	Augusto, B. D. C. (2009). *Motion Cueing in the Chalmers Driving Simulator: An Optimization-Based Control Approach.* (MSc Thesis, Chalmers University of Technology, Sweden).
[10]	Reid, L. D., & Nahon, M. A. (1985). *Flight Simulation Motion-Base Drive Algorithms: Part 1 – Developing and Testing the Equations*. (UTIAS Report 296)
[11]	Wang, Y., & Boyd, S. (2010). *Fast Model Predictive Control Using Online Optimization.* IEEE Transactions on Control Systems Technology, 18(2), 267-278. doi:10.1109/TCST.2009.2017934
[12]	Boyd, S., & Vandenberghe, L. (2004). *Convex Optimization*. doi:10.2277/ 0521833787
[13]	Gordon, T. J. & Best, M. C. (2007) *On the synthesis of Driver Inputs for the Simulation of Closed-loop Handling Manoeuvres.* International Journal of Vehicle Design, v40, pp52-76
[14]	Lehmann, E.L. (1975) *Nonparametrics: Statistical Methods Based on Ranks.* McGraw-Hill
[15]	Garrett, N.J.I. & Best, M.C. *Evaluation of a new body sideslip-based driving simulator motion cueing algorithm.* Proc. IMechE pt D: Journal of Automotive Engineering, In Press
[16]	British Formula 3 Series – http://www.formula3.co (accessed July 2012)
[17]	Noether, G.E. (1991) *Introduction to Statistics – The Nonparametric Way.* Springer-Verlag

Appendix A – Nomenclature

$\boldsymbol{A,B,C,D}$	State-space matrices
$\boldsymbol{A}$	Vector from ground origin to actuator lower joint
$\boldsymbol{A}$, $\boldsymbol{A_{eq}}$	QP constraint matrices
$\boldsymbol{B}$	Vector from platform centroid to actuator upper joint
$\boldsymbol{H}$	QP matrix
H_p, H_u	Prediction horizons
J	Cost function
K	Gain
$\boldsymbol{L}$	Rotation matrix
$\boldsymbol{Q}$	State cost coefficient matrix
$\boldsymbol{R}$	Input cost coefficient matrix
$\boldsymbol{R_{AB}}$	Vector from ground origin to platform centroid
T_s	Sample time
a	Acceleration
$\boldsymbol{b}$, $\boldsymbol{b_{eq}}$	QP constraint vectors
g	Acceleration due to gravity
k	Discrete time variable; Actuator velocity coefficient
l	Actuator length
n_x	Number of model states
p	Roll velocity
$\boldsymbol{r_p}$, $\boldsymbol{r_d}$	QP primal & dual residuals
s	Newton step multiplier
u	Control input
v	Actuator velocity
$\boldsymbol{x}$	State vector; Actuator displacement
$\boldsymbol{x_{ref}}$	Reference state vector
$\boldsymbol{z}$	QP optimization variable
Λ	Barrier function
θ	Pitch angle
κ	Barrier function coefficient
$\boldsymbol{v}$	QP dual variable
τ	Vestibular model time constant
φ	Roll angle
ψ	Yaw angle
ω	Angular velocity
$_{car}$	Simulated vehicle quantity
$_{f}$	Terminal cost parameter
$_{mpc}$	MPC linear model parameter
$_{oto}$	Otolith model parameter
$_{plat}$	Platform quantity
$_{ref}$	Reference state
$_{scc}$	Semicircular canal model parameter
$_{sim}$	Simulator quantity
$_{vest}$	Vestibular model parameter
$\hat{}$	Perceived/estimated quantity

Appendix B – Numerical values

T_s	0.025 s	MPC controller sampling time
H_p	$5 \times T_s$	MPC prediction horizon
H_u	$1 \times T_s$	MPC control horizon
τ_a	30 s	Semicircular canal time constants
τ_1	0.1 s	
τ_2	6.1 s	
K_{oto}	0.4	Otolith time constants
τ_n	13.2 s	
τ_L	5.33 s	
τ_S	0.66 s	
$Q_{perc,p}$	100	Roll perception cost
$Q_{perc,y}$	1	Lateral perception cost
$\boldsymbol{Q_{plat}}$	$1*\mathbf{1}_6$	Actuator excursion cost
R_p	0.1	Roll input cost
R_{ay}	10	Lateral input cost
K_{plat}	1000	Platform cost coefficient
K_{input}	10	Input cost coefficient
α	0.1	Backtracking line search parameters
β	0.8	

Appendix C – List of figures