2016 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
MODELING & SIMULATION, TESTING AND VALIDATION (MSTV) TECHNICAL SESSION
AUGUST 2-4, 2016 – NOVI, MICHIGAN

PREDICTIVE DISPLAYS FOR HIGH LATENCY TELEOPERATION

Mark J. Brudnak, Ph.D.
Physical Simulation & Test
U.S. Army Tank Automotive Research Development and Engineering Center (TARDEC)
Warren, MI

ABSTRACT

This paper presents a method to mitigate high latency in the teleoperation of unmanned ground systems through display prediction and state estimation. Specifically, it presents a simulation environment which models both sides of the teleoperation system in the laboratory. The simulation includes a teleoperated vehicle model to represent the dynamics in high fidelity. The sensors and actuators are modeled as well as the communication channel. The latency mitigation approach is implemented in this simulation environment, which consists of a feed-forward vehicle model as a state estimator which drives a predictive display algorithm. These components work together to help the operator receive immediate feedback regarding his/her control actions. The paper contains a technical discussion of the design as well as specific implementation. It concludes with the presentation of some experimental data which demonstrate significant improvement over the unmitigated case.

INTRODUCTION

One of TARDEC's top objectives is to lead the Army and DoD Ground Vehicle Community in the research, development, engineering, demonstration and fielding of Unmanned Ground Vehicle (UGV) systems. Teleoperation is a near-term technology which has the potential to be a quick-win for UGVs. It is and has already been employed in the cases of small UGVs (S-UGV) performing Counter Improvised Explosive Device (C-IED) and Explosive Ordinance Disposal (EOD) missions. In these cases the speeds are low and the operator is usually in close proximity to the S-UGV yielding low latency and responsive control. Long-distance teleoperation on the other hand, introduces significant latency which degrades the operator's ability to drive/control the vehicle. As such, the mitigation of this latency is the most fundamental challenge to achieving teleoperation under high latency. Consider the "basic" teleoperation configuration in Figure 1. Under low-latency the human operator functions as they would in an actual vehicle, however, under latencies of more than 200-300 ms, the operator is forced to change his/her strategy to a "move and wait" approach, which significantly degrades performance and lowers the achievable top speed.

This paper presents a method to mitigate high latency through display prediction and state estimation. This approach leverages and extends techniques developed for long-haul integration of hardware in the loop (HITL) systems over the Internet [1]. Specifically, it presents a simulation environment which models both sides of the teleoperation system in the laboratory. The simulation includes teleoperated vehicle model to represent the dynamics in high fidelity. The sensors and actuators are modeled as well as the communication channel. The latency mitigation approach is implemented in this simulation environment, which consists of a feed-forward vehicle model as a state estimator which drives a predictive display algorithm. The state estimator consists of a reduced order model which seeks to achieve an immediate estimate of how the vehicle will respond to the commanded inputs. It furthermore contains a correction term which tracks the relevant vehicle states over the long-term. The predictive display uses perspective transformation techniques to predict what a camera would see from a different location given an existing frame. These components work together to help the operator receive immediate feedback regarding his/her control actions

This paper presents the overall architecture of the mitigation approach as well as development of the mathematical algorithms used for the state estimator and predictive display. It continues with a description of the simulation implementation of the approach and finishes with

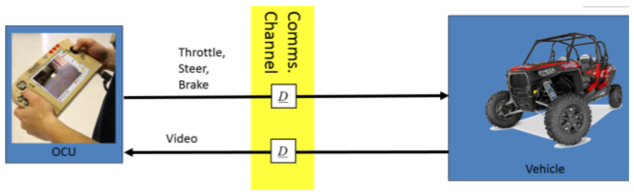

Figure 1. Basic teleoperation configuration with unmitigated delay, $\underline{D}$.

experimental comparisons of the state estimator and predictive display performance as compared to both the ideal and the unmitigated cases.

BACKGROUND

The negative impact of delay on the performance of teleoperated UGVs is well established in the literature [2], [3]. Delays in closed-loop control systems are one of the well-known sources of degraded performance and/or stability. Although, in the case of teleoperation, the human-in-the-loop is a stabilizing factor and provides robust compensation against instability, the human's ability to respond to the visual inputs begins to degrade at delays above 50ms and performance is so degraded at 200-300 ms that the operator must change his/her control approach from continuous steer to the slower and more error prone "move and wait." The "move and wait" approach requires the operator reduce speed to mitigate against the delay.

Various approaches have been used to mitigate high latency. TARDEC first encountered this problem as they were developing their duty cycle experiments [4], [5], [6]. In this approach, they used a high-fidelity model to serve as a state predictor to estimate the system's response in the future. Subsequently, TARDEC, along with the University of Michigan, began to develop additional methods of quantification of performance [7], [8], [9] and independence from and explicit model [10], [11], [12], [13], [14], [15]. These methods have transitioned to the problem of teleoperation in the presence of substantial time delays [16]. Related to the problem of predicting system behavior in the presence of delays, is the challenge of incorporating the prediction into the control algorithm. Some methods strive to place the predictors in-line either predicting a future state of the system or of the operator [16]. Since the standard teleoperation scenario incorporates a video feed being sent back to the operator, this is still delayed by the amount of transport delay between the vehicle and the operator. Furthermore, the video's data, being a series of raster images, is not subject to explicit prediction because its values are not the result of a natural evolution of system states. In this case, researchers have undertaken methods to present information in the video stream to help the operator understand the true state of the system. In these cases, many have undertaken to overlay the display with graphics to include vehicle surrogates and lane markers [17], [18]. Others have undertaken to physically manipulate the video frames to estimate what the driver would see if the stream were not delayed. Lovi, et al. have developed methods for fixed base manipulators in a manufacturing environment [19]. Rachmielowski, et al. have developed similar methods [20]. Royer, et al. used vision systems to localize a mobile robot [21]. Cobzas, et al. have used predictive displays to estimate and reconstruct geometry [2]. Kelly, et al. developed a predictive display approach using vision and LIDAR to fully construct a 3D scene including geometry and vision [22].

This work seeks to develop a state estimator and predictive display system which (1) requires minimal intervention on the vehicle and (2) which is as simple as possible. Furthermore, it seeks to present feedback to the operator in the most natural way. The predictive display approach is predicated on the assumption that a vehicle camera scene at time t is very similar to a prior scene (i.e. at time $t - 2\underline{D}$). This approach seeks to predict a current view from a past view using a predictive model in order to adjust the vantage point. This paper first discusses the mathematics behind the approach. It then discusses the simulation environment setup to evaluate it. It presents some experimental data and finishes with conclusions.

APPROACH

The top-level approach to developing this system is illustrated in Figure 2. Building upon the fundamental elements shown in Figure 1, the added components are the State Estimator (SE) and the Predictive Display (PD).

The SE functions in two modes simultaneously: feedforward and feedback. In feedforward mode the SE accepts the driver commands in the form of throttle, brake and steer (T, B, S) and (using the current state) predicts an immediate response. Since the SE runs at 100Hz, it has a very responsive reaction to the driver commands, on the order of 10 ms. This feedforward mode is based on a very simplified form of the vehicle dynamics as it is understood that high-fidelity is not needed because (1) terrain information is not known and (2) its prediction horizon is on the order of the round trip delay, $2\underline{D}$. If it were only operating in feedforward mode the states would drift from the actual values, so there is a correction term which keeps the states roughly in-line with the states of the vehicle. Finally, the SE maintains a record of states so that it can look back to a past state to obtain the relative motion between the two.

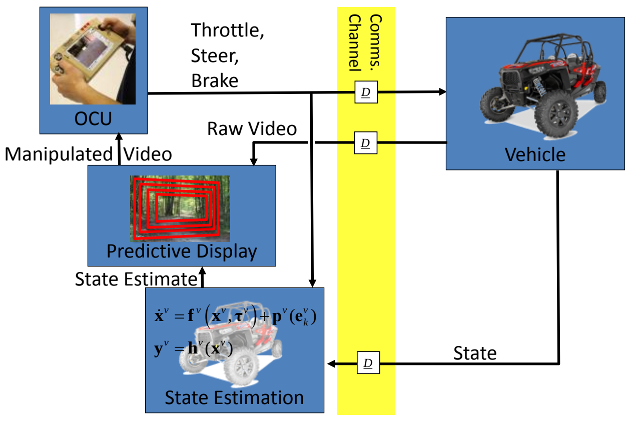
OCU
Throttle, Steer, Brake
Comms. Channel
$\underline{D}$
Vehicle
Raw Video
Manipulated Video
Predictive Display
State Estimate
$\dot{\mathbf{x}}^v = \mathbf{f}^v(\mathbf{x}^v, \boldsymbol{\tau}^v) + \mathbf{p}^v(\mathbf{e}_k^v)$
$\mathbf{y}^v = \mathbf{h}^v(\mathbf{x}^v)$
State Estimation
State

Figure 2. Block diagram of predictive display and state estimation approach.

The PD consumes the predicted state information from the SE as well as the raw video stream from the vehicle. The PD operates under the assumption that the scene observed at time *t* will be very similar to that observed at *t*-2$\underline{D}$. It therefore asks the SE to give the difference in position from time *t*-2$\underline{D}$ to time *t*. It then uses this information to manipulate the latest video frame (which is 2$\underline{D}$ seconds old) to give a best estimate of what the operator would see as if there were no delay. This then is passed to the OCU for display to the operator.

STATE ESTIMATOR

In this section the state estimator (SE) is derived. The coordinates for the state estimator are illustrated in Figure 3. The State Estimator is a planar model with three degrees of freedom defined by $\boldsymbol{p}_k = [x_k \quad y_k]^T$ and θ_k, where time is discretized with the index *k*. In this system, the state is represented as $\boldsymbol{x}_k = [\boldsymbol{p}_k^T \quad \theta_k]^T$. These are stored in the global coordinate system denoted by the subscript *A*. The local coordinate system, denoted by *B*, is used to update the rate states which are then translated into the global frame for integration. The rates are denoted as $\dot{y}_B$ and $\dot{\theta}$. The state estimator feedforward dynamics consist of a longitudinal model and a lateral/yaw model. The inputs to the vehicle coming from the OCU are denoted as throttle, t_k, brake, b_k, and steer, s_k, and are passed as unitless values normalized to a maximum of 100. The equations of motion are modeled as continuous differential equations and then discretized using the Euler approximation. The acceleration equations are as follows:

$$\ddot{y}_B = \frac{1}{M}\left(f_p(t_k, \dot{y}_B) + f_r(b_k, \dot{y}_B) + f_g(\chi)\right) \qquad (1)$$

and

$$\ddot{\theta}_B = \frac{1}{I} f_s(s_k, \dot{y}_B). \qquad (2)$$

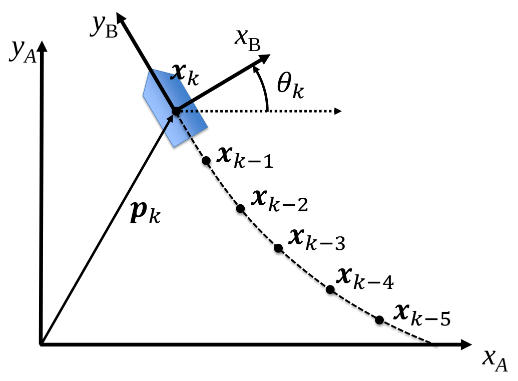

Figure 3. Coordinate system of the State Estimator.

These accelerations are integrated to obtain velocity states $\dot{y}_B$ and $\dot{\theta}_B$, which are then translated into the global frame using the rotation matrix $\mathbf{R}_A^B$ yielding

$$\dot{\boldsymbol{p}}_A = \mathbf{R}_A^B \begin{bmatrix} 0 \\ \dot{y}_B \end{bmatrix}. \qquad (3)$$

These are integrated to yield the new global position $\boldsymbol{x_A}$ and $\theta_{\boldsymbol{A}}$ which are then stored (along with the corresponding time t_k) in a circular buffer for later use. The corresponding homogeneous matrix $\mathbf{H}_A^B = \begin{bmatrix} \mathbf{R}_A^B & \boldsymbol{p}_k \\ \mathbf{0}^T & 1 \end{bmatrix}$ represents the combination of the rotation and translation of the local coordinate system, *B*.

Longitudinal Dynamics

The longitudinal dynamics, governed by $f_p(t_k, \dot{y}_B)$, $f_r(b_k, \dot{y}_B)$, and $f_g(\chi)$, determine the instantaneous speed of the vehicle. Most problems with teleoperation over a high-latency communications channel have to do with over-correction of steer direction and not speed, mostly because the vehicle is less sensitive to throttle and brake than it is to steer. That being said, because turning rate is dependent on speed, it is important that the speed be accurate to assure good yaw rate estimates. The equations for the longitudinal model use the illustration in Figure 4. There are three components of the

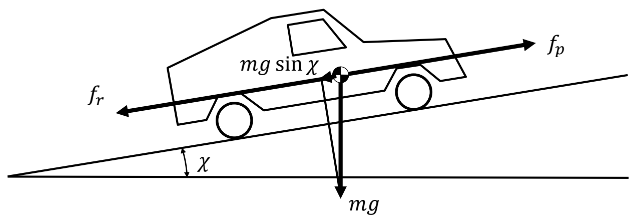

Figure 4. Forces governing the longitudinal degree of freedom.

force which act longitudinally. First f_p accounts for the propulsion force which is dependent on the throttle input and the velocity of the vehicle. With the goal of being as simple as possible, the following equation is used for the propulsion forces

$$f_p = MA_{max} \frac{|V_{max} - \dot{y}_B|}{V_{max}} \frac{t_k}{100}, \qquad (4)$$

where M is the vehicle mass, A_{max} is the maximum acceleration available to the vehicle, and V_{max} is the maximum velocity of the vehicle and $t_k \in [0,100]$. Note that this is intentionally designed to be governed by a few high-level parameters. Likewise the forces which oppose motion are lumped into f_r and these include the braking and drag. These are again modeled as simply as possible

$$f_b = \begin{cases} -MB_{max}\,\mathrm{sign}\,\dot{y}_B \frac{b_k}{100}, & |v| > 0.1 \\ 0, & \text{otherwise} \end{cases}, \tag{5}$$

where M is the vehicle mass, B_{max} is the maximum braking acceleration, $b_k \in [0,100]$ is the brake command, and $\mathrm{sign}\, v = \frac{v}{|v|}, v \neq 0$ returns the sign of its argument. The drag forces are

$$f_d = -\frac{A_f C_d}{2} \rho |\dot{y}_B| \dot{y}_B, \tag{6}$$

where A_f is the cross sectional area, C_d is the drag coefficient, and ρ is the density of air.

Finally gravity component due to grade is

$$f_g = -MG \sin \chi, \tag{7}$$

where G is the acceleration due to gravity (i.e. 9.8 m/s^2), χ is the pitch angle of the vehicle.

Lateral/Yaw Dynamics

Lateral/yaw dynamics are illustrated in Figure 5. As discussed, it is more important for this to be accurate since a vehicle is typically much more responsive to steering inputs than to longitudinal inputs. Based on the kinematics shown in the figure, it is reasonable to use $\dot{\theta} = \frac{\dot{y}_B}{B} \tan \varphi$, however, the form shown in equation (2) is desirable because the correction term can then work through an integrator rather than directly on the state. In that case

$$f_s = \frac{I}{\Delta t}\left(\frac{\dot{y}_B}{B} \tan\left(\beta \frac{s_k}{100}\right) - \dot{\theta}\right) \tag{8}$$

where I is the yaw moment of inertia (note that the equations are structured such that I does not matter), B is the wheel base, β is the steering scale factor which accounts for gain and conversion to radians, $s_k \in [-100, 100]$ is the steer command, $\dot{\theta}$ is the current yaw rate, and Δt is the time step of the dynamics model. (Note that the relationship to Figure 5 is that $\varphi = \beta \frac{s_k}{100}$.)

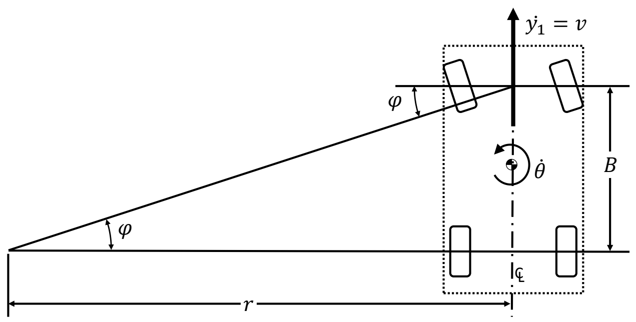

Figure 5. Illustration of the lateral/yaw estimator.

State Correction

Regardless of the accuracy of the feedforward model, states tend to drift. It is therefore necessary to add a feedback term. As such, the two terms that are subject to correction are the speed, $\dot{y}_B$, and the yaw rate, $\dot{\theta}$. First note that the design employs remote error computation. In this method, the estimated state is sent to the vehicle along with the vehicle commands. The estimated state is then compared to the actual state on the vehicle and the error is sent back to the OCU. In this way, the error computation is uninfluenced by time skew. This approach is illustrated in Figure 6(b) as compared to the skewed computation (Figure 6(a)). The error is then used to close the gap between the estimated and measured states, understanding that it is 2$\underline{D}$ seconds old. The rate of correction must account for the arrival of new error information every 10 ms (i.e. the state estimator runs at 100 Hz). The correction gain is therefore set so that the error will be closed in about 100ms. Equations (1) and (2) then become

$$\ddot{y}_B = \frac{1}{M}\left(f_p(t_k, \dot{y}_B) + f_r(b_k, \dot{y}_B) + f_g(\chi)\right) - g_v e_{\dot{y}_B} \tag{9}$$

and

$$\ddot{\theta}_B = \frac{1}{I} f_s(s_k, \dot{y}_B) - g_y e_{\dot{\theta}_B} \tag{10}$$

where g_v and g_y are the gains for the correction terms. In this work they are both set to 10.

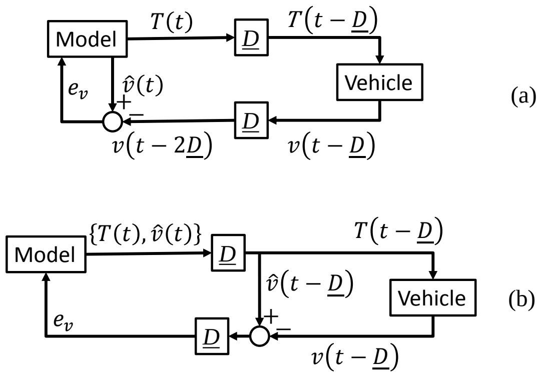

Figure 6. Two approaches to error computation. Example shown in the case of longitudinal performance where *T*(*t*) represents the throttle command and *v*(*t*) represents the velocity. The top diagram (a) represents error computation on the OCU/estimator side where the computation is skewed by 2$\underline{D}$ seconds and the bottom diagram (b) represents error computation on the vehicle side which is not skewed.

PREDICTIVE DISPLAY

The basic idea of the predictive display is illustrated in Figure 7. The top of the figure illustrates two positions of the camera mounted on the vehicle. The first position shows the video frame represented with no delay. The second position shows the camera corresponding to forward motion. The basic idea of the predictive display is that the frame is (in concept) projected onto the ground plane and the far plane. The position is then moved forward by 2$\underline{D}$ according to the speed. The image is then projected back onto a virtual image plane to account for the forward motion. This gives a reasonable estimate as to what the operator will see at a time that is advanced 2$\underline{D}$ into the future. How this works with an actual image is shown on the bottom of Figure 7. From left to right is shown the original image, the projection of the moved camera back onto the image plane, and the subsequent transformed image. Figure 7 only illustrates longitudinal motion. To fully account for planar motion, the approach incorporates lateral and yaw motion as well.

The predictive display is configured with three coordinate systems as shown in Figure 8. The global frame denoted by "0" represents the notional location of the vehicle when the video frame was grabbed. It represents the no-delay case. It is also the frame in which the ground plane and far plane remain fixed. (Note: It does not correspond to the frame A in the vehicle state estimator as shown in Figure 3.) The vehicle frame denoted by "1" represents the location of the vehicle after its position has evolved over the round-trip delay of 2$\underline{D}$. The camera frame denoted by "2" represents the location of the camera in the vehicle frame and it is fixed with respect to the vehicle (although in general it does not need to be). The frame "2′" represents frame "2" with respect to the global frame "0". Also shown in Figure 8 is the image plane and the trace of the four corners of the image plane onto the ground and far planes represented by black lines and red traces on the ground and far planes.

The camera coordinates are illustrated in Figure 9. There the image plane is located in physical units in the camera coordinate system. The image plane is translated to pixel units as a raster. (XGA resolution of 1024x768 is used in the sequel.) Conversion between raster and image plane coordinates can be found in any computer vision text such as Szeliski [23].

The steps associated with the perspective transformation are illustrated in Figure 10. These steps are described in the remainder of this section.

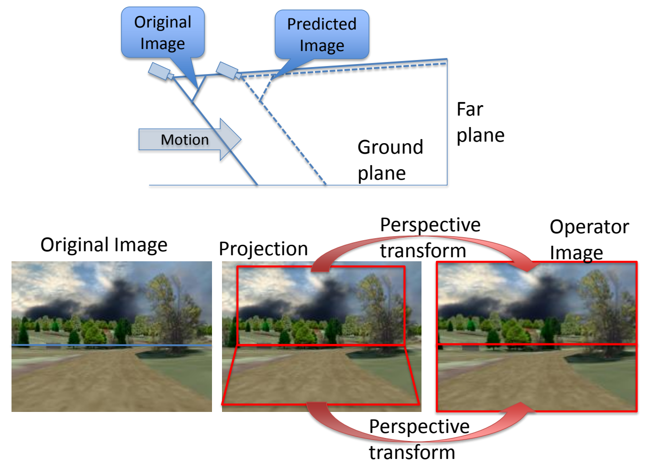

Figure 7. Illustration of the main idea of the predictive display. The top shows how the original scene is projected onto the ground plane and far plane. The bottom shows at a high-level how this information is used to manipulate the video frame.

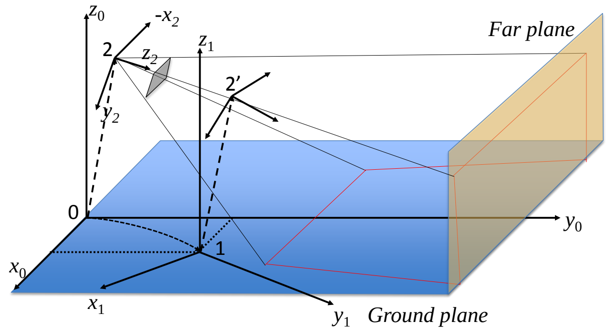

Figure 8. Definition of the three coordinate systems used for the predictive displays. The global system denoted by "0" represents the coordinates from which the original frame was captured. The vehicle coordinate system denoted by "1" (corresponds to B) represents the position of the vehicle after 2$\underline{D}$ seconds. The camera coordinate system "2" represents the location of the camera with respect to the vehicle.

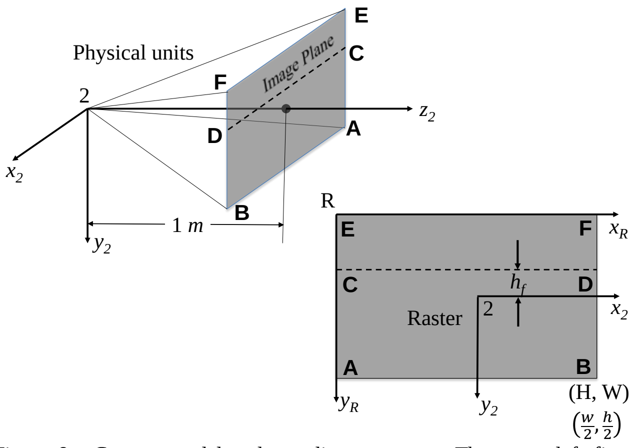

Figure 9. Camera model and coordinate system. The upper left figure illustrates the location of the image plane in the camera coordinate system. The lower-right figure illustrates the raster coordinates. The image plane and raster correspond, but are measured in physical units and pixels respectively.

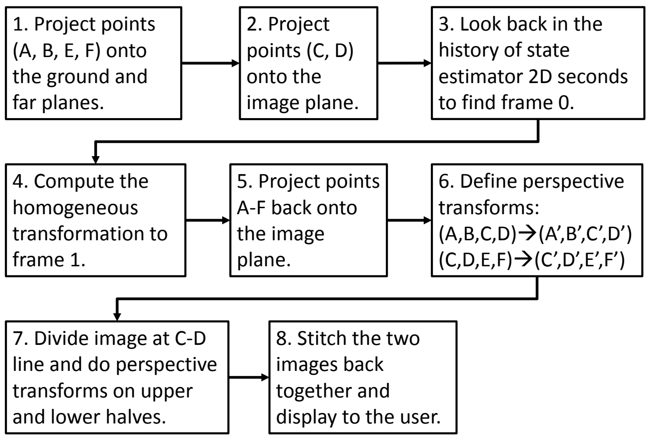
1. Project points (A, B, E, F) onto the ground and far planes.
2. Project points (C, D) onto the image plane.
3. Look back in the history of state estimator 2D seconds to find frame 0.
4. Compute the homogeneous transformation to frame 1.
5. Project points A-F back onto the image plane.
6. Define perspective transforms: (A,B,C,D)→(A',B',C',D') (C,D,E,F)→(C',D',E',F')
7. Divide image at C-D line and do perspective transforms on upper and lower halves.
8. Stitch the two images back together and display to the user.

Figure 10. Flow chart for the image transformation for the predictive display.

Step 1: Mapping of points A, B, E, F

This step may be done in preprocessing as long as the camera is fixed to the vehicle. First define camera parameters such as field of view ϑ and aspect ratio α. The height and width of the image plane are,

$$\begin{aligned} w &= 2\tan\frac{\vartheta}{2}, \\ h &= \alpha w, \\ \varphi &= 2\tan^{-1}\frac{h}{2}, \end{aligned} \tag{11}$$

where w and h are the width and height (in meters) of the image plane and φ is he vertical field of regard of the camera. Capital letters W and H are used to denote the width and height of the raster in pixels. In this case $W = 1{,}024$ and $H = 768$. Figure 11 illustrates the situation of the camera with respect to frame “1” and also illustrates the location of the far plane with respect to frame “1” when it coincides with frame “0”. Let ψ be the down angle of the camera and $\boldsymbol{p}_c$ be the position of the camera in frame “1”. Let d_f be the distance of the far plane from the origin of frame “0”. Furthermore let ε be the angle between the z_2 axis and the line traced from the origin of “2” to the intersection of the ground and far planes.

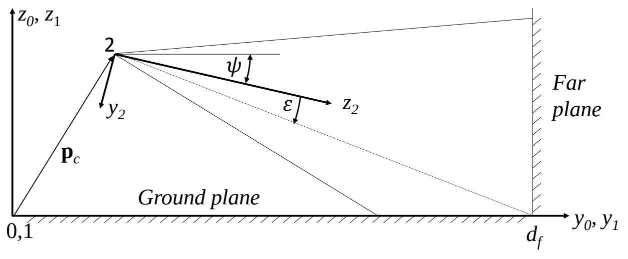

Figure 11. Side view of camera coordinate system and far plane position when fames “0” and “1” coincide.

Given these definitions, let the rotation matrix between frames “1” and “2” be

$$\mathbf{R}_1^2 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -\sin\psi & \cos\psi \\ 0 & -\cos\psi & -\sin\psi \end{bmatrix}, \tag{12}$$

and the homogeneous transformation be

$$\mathbf{H}_1^2 = \begin{bmatrix} \mathbf{R}_1^2 & \boldsymbol{p}_c \\ \mathbf{0}^T & 1 \end{bmatrix}. \tag{13}$$

The computation proceeds as follows. First, project image plane points A and B (see Figure 12) onto the ground plane. For point A, its position in the camera frame is $\boldsymbol{v}_2^A = [\frac{-w}{2} \quad \frac{h}{2} \quad 1]^T$. To map A to the ground plane, the following equation must hold for unknown γ

$$(\boldsymbol{p}_c + \gamma \mathbf{R}_1^2 \boldsymbol{v}_2^A)^T \boldsymbol{n}_g = 0, \tag{14}$$

where $\boldsymbol{n}_g$ is a vector normal to the ground plane. The solution to this equation yields $\gamma = \frac{-\boldsymbol{p}_{c,z}}{\boldsymbol{v}_{1,z}^A}$. Where the *z* in the subscript indicates that the *z* component of the vector is used. This then yields

$$\boldsymbol{v}_0^A = \boldsymbol{p}_c + \gamma \mathbf{R}_1^2 \boldsymbol{v}_2^A. \tag{15}$$

The same procedure is used for B. Next to find E and F, the point E is used here. First, let $\boldsymbol{v}_2^E = [\frac{-w}{2} \quad \frac{-h}{2} \quad 1]^T$, then the following equation must hold for unknown η

$$\left(\boldsymbol{p}_c + \eta \mathbf{R}_1^2 \boldsymbol{v}_2^E - \boldsymbol{p}_f\right)^T \boldsymbol{n}_f = 0, \tag{16}$$

where $\boldsymbol{p}_f$ is the vector to the intersection of the far and ground plane and $\boldsymbol{n}_f$ is a vector normal to the far plane. The solution to this equation yields $\eta = \frac{\boldsymbol{h}_f - \boldsymbol{p}_{c,y}}{\boldsymbol{v}_{1,y}^E}$. Where the *y* in the subscript indicates that the *y* component of the vector is used. This then yields

$$\boldsymbol{v}_0^E = \boldsymbol{p}_c + \eta \mathbf{R}_1^2 \boldsymbol{v}_2^E. \tag{17}$$

The same procedure may be applied to the point F.

Step 2: Mapping of points C & D

Unlike points A, B, E, F which were well-defined in the camera frame but unknown in the global frame, points C and D are fairly well defined in the global frame “0”, but not well defined in the image frame. First observe that they need to be on the edges of the image so their x_2 coordinate will be $\pm\frac{w}{2}$. It can easily be shown that $\varepsilon = \tan^{-1}\frac{d_f - p_{c,y}}{p_{c,z}} + \psi - \frac{\pi}{2}$ (see

Figure 11). It is straightforward to calculate $h_f = \tan\varepsilon$ (see Figure 9). Then, let $\boldsymbol{v}_2^C = [\frac{-w}{2} \quad h_f \quad 1]^T$ and proceed with the same method outlined in equations (14) and (16). The same process applies to point D. This step may also be precomputed as long as the camera and far plane are not moving from time step to time step.

Step 3: Determine position evolution over 2D

This step recalls a prior state from the buffer so that a relative position between the current and prior state may be computed in the next step. The states stored in the buffer take the following form $[x_k \quad y_k \quad \theta_k \quad t_k]$ which includes the 3 degree of freedom position and the time at which the state was computed. Assuming that the round trip time is known, which is denoted as τ, the algorithm looks back in the buffer i steps until $t_k - t_{k-i} \geq \tau$. When this condition is met, let $j = k - i$.

Step 4: Compute the transformation from frame 0 to 1

These two states (at sample k and j) are then encoded as homogeneous transformations as

$$\mathbf{R}_{A\,k,j}^{B} = \begin{bmatrix} \cos\theta_{k,j} & -\sin\theta_{k,j} & 0 \\ \sin\theta_{k,j} & \cos\theta_{k,j} & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad (18)$$

$$\boldsymbol{p}_{k,j} = [x_{k,j} \quad y_{k,j} \quad 0]^T, \qquad (19)$$

$$\mathbf{H}_{A\,k,j}^{B} = \begin{bmatrix} \mathbf{R}_{A\,k,j}^{B} & \boldsymbol{p}_{k,j} \\ \mathbf{0}^T & 1 \end{bmatrix}, \qquad (20)$$

where the indices k, j indicates that the computation is made for k and j independently. The instantaneous transformation between frames "0" and "1" is then

$$\mathbf{H}_{0\,k}^{1} = \mathbf{H}_{A\,j}^{B\;-1} \mathbf{H}_{A\,k}^{B} \qquad (21)$$

Step 5: Project points A-F back onto image plane.

For each of the points A-F and their corresponding global positions $\boldsymbol{v}_0^A, \ldots, \boldsymbol{v}_0^F$ computed in steps 1 and 2, they are translated back into the camera frame as follows

$$\begin{bmatrix} \boldsymbol{v}_2^A \\ 1 \end{bmatrix} = \mathbf{H}_1^{2\;-1} \mathbf{H}_{0\,k}^{1\;-1} \begin{bmatrix} \boldsymbol{v}_0^A \\ 1 \end{bmatrix}. \qquad (22)$$

This yields the corresponding points A′-F′ in the camera frame "2", which are then converted to planar coordinates as follows

$$\boldsymbol{u}^A = \begin{bmatrix} \boldsymbol{u}_x^A \\ \boldsymbol{u}_y^A \end{bmatrix} = \begin{bmatrix} \boldsymbol{v}_{2,x}^A / \boldsymbol{v}_{2,z}^A \\ \boldsymbol{v}_{2,y}^A / \boldsymbol{v}_{2,z}^A \end{bmatrix}, \qquad (23)$$

where $\boldsymbol{v}_2^A = \left[\boldsymbol{v}_{2,x}^A \quad \boldsymbol{v}_{2,y}^A \quad \boldsymbol{v}_{2,z}^A\right]^T$. Note that it is important to place the far plane such that $d_f > \boldsymbol{p}_{c,y} + v_{max}\tau_{max}$, where v_{max} is the maximum velocity and τ_{max} is the maximum round trip time so that the possibility of $\boldsymbol{v}_{2,z}^A = 0$ is physically impossible. Finally once the desired points are obtained in physical coordinates, the last step is to convert them to raster coordinates (i.e. pixels) as follows.

$$r_x^A = u_x^A \frac{\mathrm{W}}{w} + \frac{\mathrm{W}}{2}, \qquad (24)$$

$$r_y^A = u_y^A \frac{\mathrm{H}}{h} + \frac{\mathrm{H}}{2}. \qquad (25)$$

where capitals (W, H) represent the size in pixels and lower case (w, h) are physical units. The same procedure is used for points B-F respectively. This process is illustrated in Figure 12. There the image plane is extended to permit the projected points to fall outside of the bounds of the image. Figure 13 illustrates how the points move on the image plane

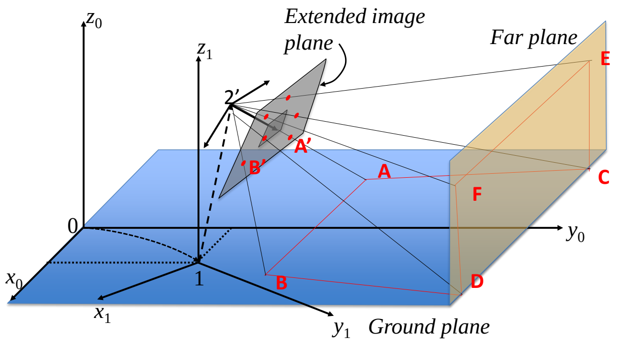

Figure 12. Points A through F are projected back from global frame "0" after the vehicle's movement represented by frame "1". The point A maps to A′, B to B′, etc. These points do not have to land in the actual image for the transformation to work. To reduce clutter, only points A′ and B′ are shown.

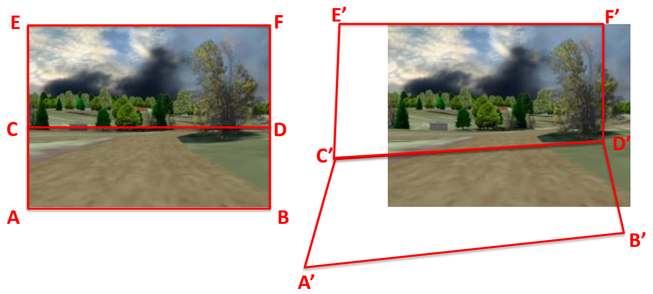

Figure 13. Mapping of points A-F to A′-F′. Note that they will not typically land back within the boundaries of the original image. These mappings will be used to define two perspective transformations.

Step 6: Define perspective transforms

In this step two perspective transforms are defined, one for the ground plane which is uniquely defined by the mapping (A, B, C, D)→ (A', B', C', D') and one for the far plane (C, D, E, F)→ (C', D', E', F'). The perspective transform for the ground plane has the following form

$$\begin{bmatrix} a\boldsymbol{r}^{A'} & b\boldsymbol{r}^{B'} & c\boldsymbol{r}^{C'} & d\boldsymbol{r}^{D'} \\ a & b & c & d \end{bmatrix} = \mathbf{M}_{GRD} \begin{bmatrix} r^{A} & \boldsymbol{r}^{B} & \boldsymbol{r}^{C} & \boldsymbol{r}^{D} \\ 1 & 1 & 1 & 1 \end{bmatrix}, \quad (26)$$

and for the far plane,

$$\begin{bmatrix} e\boldsymbol{r}^{C'} & f\boldsymbol{r}^{D'} & g\boldsymbol{r}^{E'} & h\boldsymbol{r}^{F'} \\ e & f & g & h \end{bmatrix} = \mathbf{M}_{FAR} \begin{bmatrix} r^{C} & \boldsymbol{r}^{D} & \boldsymbol{r}^{E} & \boldsymbol{r}^{F} \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad (27)$$

where $\mathbf{M}_{GRD} \in \mathbb{R}^{3\times 3}$, $\mathbf{M}_{FAR} \in \mathbb{R}^{3\times 3}$ and $a, \dots, h$ are arbitrary constants. The perspective transform is computed using the image processing library *OpenCV* [24]. Specifically the function `getPerspectiveTransform()` is used to compute the transform for both the ground and far plane.

Step 7: Split the image and transform each half.

In this step the image is split along the C-D line, with that below associated with the ground plane and that above associated with the far plane. These two sub-images are then transformed using the learned perspective transforms $\mathbf{M}_{GRD}$ and $\mathbf{M}_{FAR}$. (Note that care must be taken to make sure that the mapped points correspond to the divided image.) This is done using the *OpenCV* `warpPerspective()` function. This function maintains the bounds of the original image.

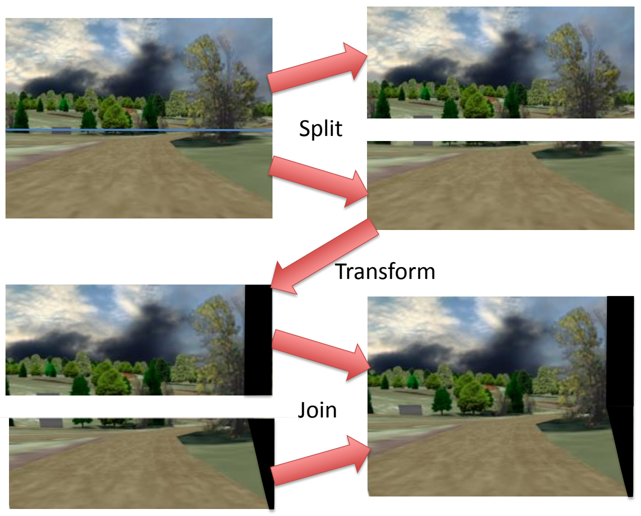

Figure 14. Steps 7 & 8 shows splitting the image, performing the perspective transform and then rejoining the image. Illustrated here is a perspective transform indicated by a forward turn to the right. Notice that the black areas represent pixels for which there is no source information.

Clearly two things happen as illustrated in Figure 13. First there are pixels which map outside of the bounds of the original image; these points are lost in the destination image. Second, there are pixels in the destination image which are sourced outside of the bounds of the original image. In this case, *OpenCV* paints these pixels black.

Step 8: Join the transformed images

Finally take the two sub-images and rejoin them and present them to the user. This step is illustrated in Figure 14.

SIMULATION SETUP

The above design was implemented on two workstations running Microsoft Windows 7 Professional 64 bit as shown in Figure 15. They were connected via a gigabit Ethernet LAN and passed all relevant information via the UDP/IP protocol. Although these computers have the power to run everything on one machine, they were run on two for a few reasons. First, because this is a simulation, information is very accessible. By running them on two different machines, it is assured that only valid/relevant state is being shared (i.e. no access to privileged information). The second is that because information is passed over a physical network, it provides well-defined "choke points" to monitor and/or control network behavior. This gives ample opportunity to control network performance via packet forwarding or the insertion of a network emulator. In the following sections, the Vehicle Sim and OCU Sim software design are described.

Vehicle Sim

The Vehicle Sim computer implemented the majority of its software in *SimCreator* 3.0 which is a tool designed to integrate simulation components using block diagrams [25], [26], [27] and model multi-body dynamics. *SimCreator* enables simulations to be distributed both on a computer and across a LAN. In this case it was configured to run three processes on the same machine as shown in Figure 16. Information is passed between these processes by *SimCreator* using the UDP/IP protocol on the machine.

The vehicle process did just as it says, it ran a real-time vehicle dynamics model. The model was built in *SimCreator* as shown in Figure 17. The model includes full suspension

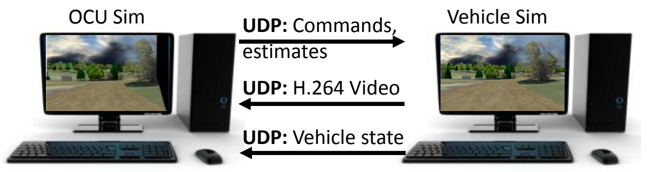

Figure 15. Setup of the computers on which this long haul teleoperation was configured. All of the state estimation and predictive display software as well as the user interface ran on the OCU Sim. The vehicle dynamics, image generator and video encoder ran on the Vehicle Sim computer. The computers were connected to a Gigabit Ethernet LAN.

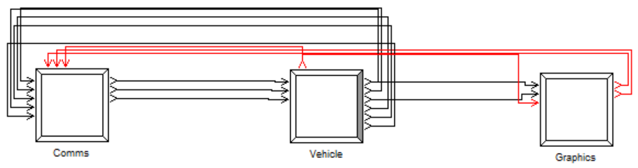

Figure 16. Blocks representing three processes for the Vehicle Sim. *Comms* performed UDP communication and network delay modeling, *Vehicle* contained the multibody vehicle dynamics of the HMMWV used for the UGV, and *Graphics* ran the graphical rendering engine and communicated with another process via shared memory which ran the video encoding.

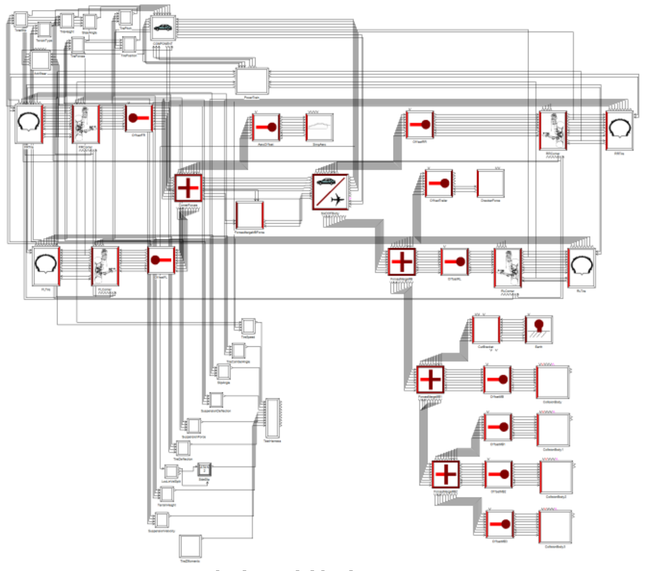

Figure 17. HMMWV vehicle model built in *SimCreator*.

kinematics, power train and tire model. It also communicates with a terrain model for roughness, grade and side slope information. It is updated at 1,000 Hz and its states are integrated using Runge-Kutta 4th order integrator. It receives driver commands from the Comms process and provides vehicle state information (to include position, velocity and acceleration as well as orientation and rotational rates) to the Comms and Graphics processes.

The Comms process manages UDP communications to the OCU Sim computer. It receives UDP packets from the OCU Sim according to the structure shown in Listing 1. In that structure is a monotonically increasing packet number and a time stamp. The time stamp is sent back by the vehicle so the OCU can estimate round trip time. Vehicle commands include throttle, brake, steer and gear. Finally, the estimated states of speed and yaw rate are sent to the vehicle for computation of the error. The OCU Sim generated these packets at 100 Hz. This received `OcuToVehicle` packet is a good place to insert a network performance model and the Comms process does just that by enqueuing these packets in a FIFO buffer along with their associated time stamps. They are released from the FIFO queue when they have been waiting for at least τ seconds.

When released from the FIFO queue, the Comms process responds by immediately sending out a packet with the structure shown in Listing 2. This contains the received packet number and time stamp as well as the local time stamp. This is used to synchronize time between the two machines. The video frame number and video frame time are sent for tracking purposes. It contains the orientation (only pitch is used), the angular rates (only yaw rate is used) and speed. Finally, it contains the errors which were computed based on the sent speed and yaw rate estimates. It was previously established that computing error on the vehicle side is a simple and effective way to assure that it is not affected by time skewing of the signals.

The Graphics process renders the view that the camera sees. It takes in a terrain database model in *Open Flight* or *VRML* format and then renders the view based on position and orientation, which it receives from the Vehicle process. It

```
struct OcuToVehicle
{
  // General information
  int packetNo; // Incremental count
  unsigned int ocuTime; // Time stamp

  // Vehicle commands
  float throttle; // [0,100]
  float brake; // [0,100]
  float steer; // [-100,100]
  int gear; // Enumerated {1,2,3,4}

  // Estimated states
  float vHat ; // Estimated speed in m/s.
  float yawRateHat; // Estimated yaw rate in rad/s.
};
```

Listing 1. Information sent from OCU to Vehicle.

```
struct VehicleToOcu
{
  // Synchronization information
  int retPacketNo; // Returned packet number.
  unsigned int retOcuTime; // Returned time.
  unsigned int vehTime; // Vehicle local time.

  // Video information
  int lastFrame; // Number of last frame.
  int frameTime; // Time of last frame.

  // IMU Data
  float orientation[3]; // R, P, Y, rad
  float omega[3]; // Rotational rates, rad/sec
  float speed; // Speed in m/s, body fixed

  // Computed errors
  float eSpeed; // Speed error
  float eYawRate; // Yaw rate error
};
```

Listing 2. Information sent from Vehicle to OCU.

renders a scene at a rate of 30 Hz at XGA resolution. These frames are rendered both on the screen and to a bit map for video encoding by a separate process.

For reasons beyond the scope of this paper, the video encoder was written outside of *SimCreator*. It is coded in C++ as a stand-alone process which communicates with the Graphics process via shared memory (using the *Boost C++ Library* [28]). It uses the *FFmpeg* [29] video encoding/decoding library to encode the video frames being passed over the shared memory. The H.264 encoder is used with the parameters shown in Table 1. For every frame output by the encoder, the information is manually chopped into 1,300-byte chunks and then sent via a UDP socket to the OCU computer. (Note that the *FFmpeg* `avio_write()` was not used because it only sends full UDP packets.)

Table 1. FFmpeg H.264 Encoder Parameters.

GOP Size	25
Bit rate	4,000,000
Resolution	1024x768
Time Base (N/D)	1/30
Max B Frames	0
Option: "tune"	"zerolatency"

OCU Sim

The OCU Sim computer runs in a single process written in C++. It employs the *FFmpeg* library for video decoding and the *OpenCV* library for the image processing. Additional components are the state estimator and the UDP communications. The code runs in two threads. The first thread runs the *FFmpeg* decoding and the *OpenCV* image processing. This first thread is event driven. It continually decodes the video stream as packets are received and then when it has a new frame, it runs the predictive display code and then displays that to the user. The second thread is a periodic loop which runs at 100 Hz. It updates the state estimator and then sends a UDP packet containing the information in Listing 1 to the vehicle. In its idle time, it also continually polls for return packets from the vehicle. When it receives a packet it updates its internal information to be used by the state estimator and the predictive display.

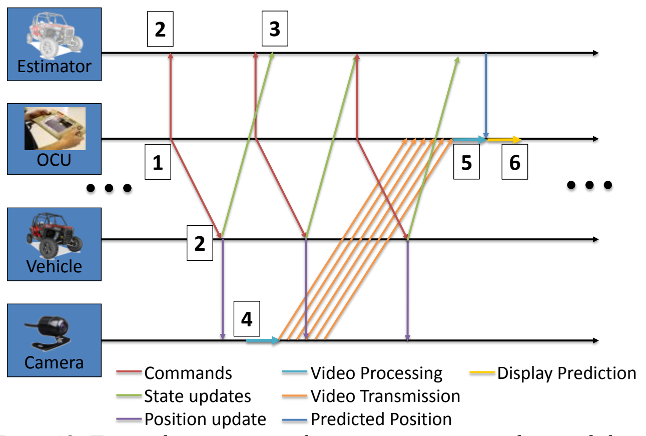

Figure 18. Timing diagram sowing the major components and a causal chain of events which start with a driver command and end with a display rendered to the operator. Also show are the flow of information between the major components. The top two blocks (Estimator & OCU) run on the OCU Sim and the bottom two blocks (Vehicle & Camera) run on the Vehicle Sim. Lines trace data flows. Time proceeds from left to right.

Timing

Figure 18 illustrates the data flow and timing of the predictive display system in a causal way. The timing and data flow are one instance of a process that occurs over and over again. The flow of information starts with a command being issued by the operator $\boxed{1}$ (which occurs at 100 Hz) which then flows to the vehicle and to the state estimator $\boxed{2}$. The arrival of the driver command immediately triggers a response in which the vehicle sends its state information and error back to the estimator $\boxed{3}$. The driver command information $\boxed{2}$ also affects the velocity and position of the vehicle and subsequently the camera $\boxed{4}$, which runs independently and captures frames at 30 Hz. Between events $\boxed{4}$ and $\boxed{5}$ the frame is encoded (Note that with GOP size of 25, most frames are P frames which require some amount of processing), chopped into UDP packets, sent to the OCU, and then reassembled and decoded into a video frame. At event $\boxed{5}$ (which is triggered by a new video frame) the OCU uses the state estimator to perform the predictive display image manipulation and then renders the frame to the operator, thus completing the cycle at event $\boxed{6}$.

EXPERIMENTAL RESULTS

The system which was described in the preceding section, was run experimentally by the author and these results are discussed in this section. The experiments were run on a flat terrain database which consists of four different types of tiles which are 200 m x 200 m each. These tiles consist of a straight section, a 75 m radius right turn, a 75 m radius left turn and an 'S' turn (which has six curves of 20 m radius each). A wire frame diagram of the terrain is shown in Figure 19. The course was negotiated in a clock-wise diection, starting on the far right (east) portion facing down (south) as shown in Figure 19. The operator's goal is to to stay in the right lane which has a width of 4 m. Speed on the course is regulated by means of speed limit signs which are as follows. Preceeding each right or left turn with 75 m radius, there is a 30 mph (48 kph) speed limit sign, preceeding each 'S' turn there is a 15 mph (24 kph) speed limit sign and preceeding each straight section of length of two tiles there is a 40 mph (65 kph) speed limit sign. The course is approximately 5.8 km (3.6 miles) long and takes approximately 10 minutes to

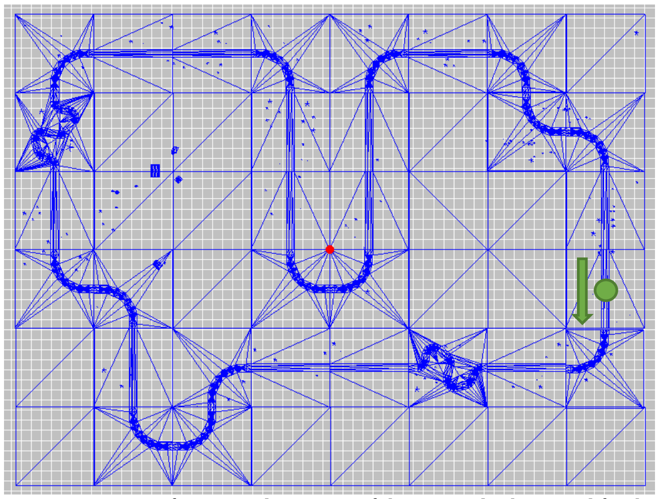

Figure 19. A wire-frame top-down view of the terrain database used for the experimental runs. The database consists of a 6 by 8 grid of 200m square tiles consisting of straight, left, right and 'S'-turn tiles. The starting location is denoted with the green circle and the direction of travel with the arrow.

complete one circuit around the course. The advantage of a tile-based terrain is that a participant is subject to the same circumstance at multiple times through the course which multiplies the statistical sample size if one wishes to examine particular events. In particular this terrain contains 14 straight sections, 9 right turns, 5 left turns, and 2 'S' turns.

The experiments consisted of three configurations as shown in Table 2. Configuration 1 with no additional latency and no predictive display is intended to represent the best possible scenario and should produce the best case. Configuration 2 on the other hand is intended to represent the baseline case where latency is present but it is not actively mitigated. This may be thoughof as the worst case baseline performance from which to improve. Configuration 3 maintains the same latency as configuration 2 however, the predictive display is added. For each configuration, five experimental runs were made. They were run in order of configuration with five of configuration 1, then five of configuration 2, etc. (Configuration 2 had one additional aborted run because the vehicle "crashed" partway through the run.)

Data collected during the experiments consisted of four data logs associated with the OcuSim and the three processes run on the VehicleSim. On the OCU the data were logged at 100 Hz and on the VehicleSim the data were logged at 1,000 Hz. Most of the results hereafter presented speak to the effectiveness of the predictive display vs. the unmitigated case. The traces of the x-y position of these runs are

Table 2. Experimental Configurations.

Configuration	Added Latency	Predictive Display
1	0 ms	No
2	500 ms	No
3	500 ms	Yes

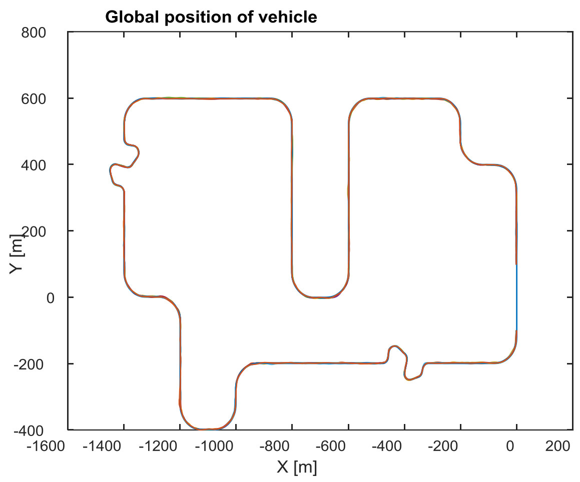

Figure 20. The path driven for all 15 experimental runs. Note the omission of analyzed data from the first tile (outlined as a dashed square).

illustrated in Figure 20. Since the section of the terrain where the simulation starts and ends has inconsistencies such as start time, stop location, etc., the data analysis omits the fist portion of the run (i.e. analysis starts with the second tile), likewise, the same tile is omitted from the end of the analysis, so when the vehicle enters this tile data analysis stops. The key metrics of interest in teleoperation are speed and accuracy where both should be maximized. (These are normally mutually opposed objectives.) In this analysis, accuracy is tracked using two metrics, namely path deviation and heading deviation (these are error metrics, so lower is better). The target path is not marked on the road but is regarded as the center of the 4 m wide right lane. The road way is defined by points along the center of the road, and the desired path is 2m to the right of

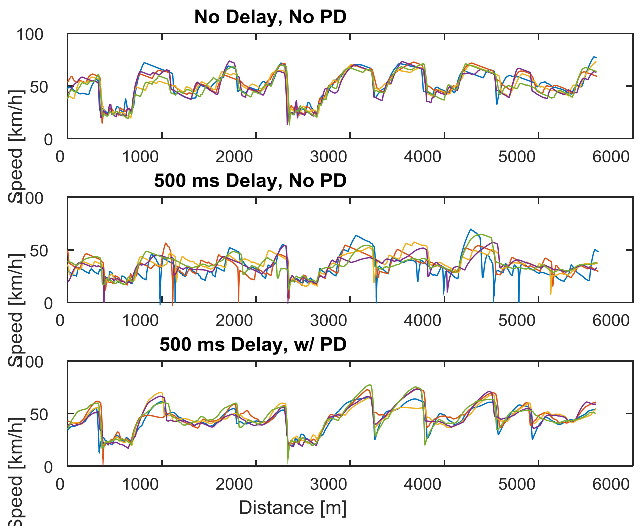

Figure 21. Speed traces for the 15 record runs vs. distance driven. Top is no delay, no Predictive Display (PD), middle is 500 ms of added delay, no PD, bottom is 500 ms of added delay with PD.

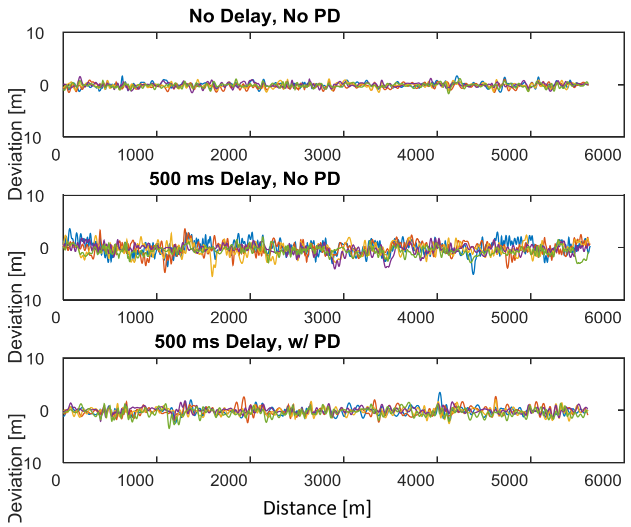

Figure 22. Path deviation traces for the 15 record runs vs. distance driven. Top is no delay, no Predictive Display (PD), middle is 500 ms of added delay, no PD, bottom is 500 ms of added delay with PD.

the center of the road. Path heading error is measured as the angle difference between the path tangent and the vehicle heading direction. Speed is measured as the average speed along the course. The raw data recorded for each of the three configurations is shown in Figure 21 for the veicle speed, Figure 22 for the path deviation, and Figure 23 for the path heading error. With respect to speed, it is clearly seen that the speed degrades severely between the first and second configuration, with the unmitigated delay case demonstrating severe inconsistency in run-to-run speeds over the course. The author believes that this is because so much attention has to be paid to steering the vehicle, that the operator pays less attention to speed. The bottom plot in Fiure 21 demonstrates that speed consistency is imporoved since the predictive display makes steering the vehicle less intense.

Figures 22 and 23 adress accuracy. Figure 22 shows path deviation which is the vehicle's distance from the center of the lane. This metric obviously directly relates to accuracy and gives an intuative sense as to how well the operator is keeping to his lane. Given that the vehicle is narrower than the lane, it is possible that the operator has a non-zero deviation but is still in the lane. Figure 23 shows the directional error as this indicates how well the operator is maintaining vehicle direction along the path. It is also an indicator of the ability (or inability) of the operator to attain and maintain a desired directional heading.

To get a qualitative sense as to how these metrics compare, the mean speeds are charted in Figure 24. There it is clear that the predictive display helps the operator achieve nearly the same average speed as the best case. In the case of configuration 2 a gradual rise in attained average speed is observed as the runs progress from run 1 to run 5. This is an indicator that an operator can adapt or train to a condition of high latency and as such improve performance. For a quantitative sense of the error, both the path deviation and the heading error are integrated over the path length (this metric is used in other TARDEC teleoperation work) as follows:

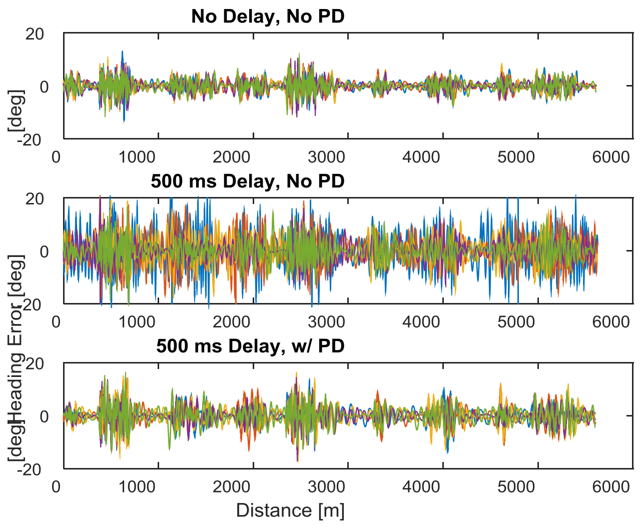

Figure 23. Path heading error traces for the 15 record runs vs. distance driven. Top is no delay, no Predictive Display (PD), middle is 500 ms of added delay, no PD, bottom is 500 ms of added delay with PD.

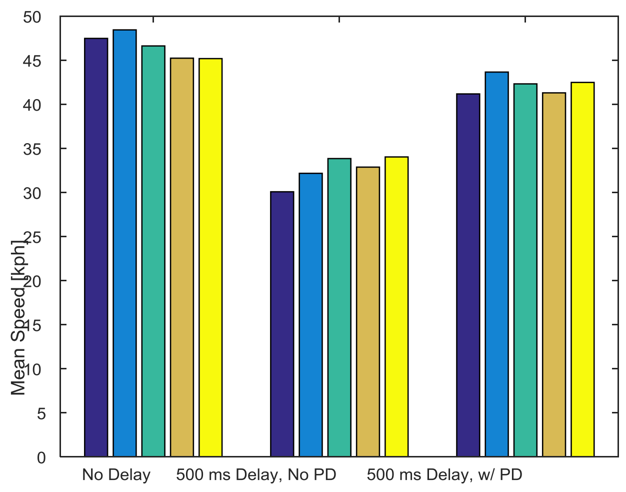

Figure 24. Mean speeds for the 15 runs. First five represent configuration 1, second five represent configuration 2, and the third represents configuration 3.

$$\int_0^L \left|e_p(s)\right| ds \tag{28}$$

where L is the length of the course under analysis (approximately 5,600 m) and $e_p(s)$ is the path deviation as a function of path length s. Likewise the metric for heading error is computed as

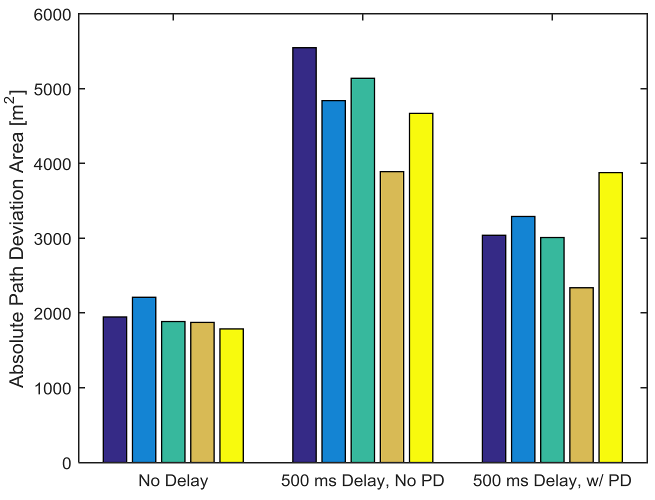

Figure 25. Integrated path deviation for the 15 runs. First five represent configuration 1, second five represent configuration 2, and the third represents configuration 3.

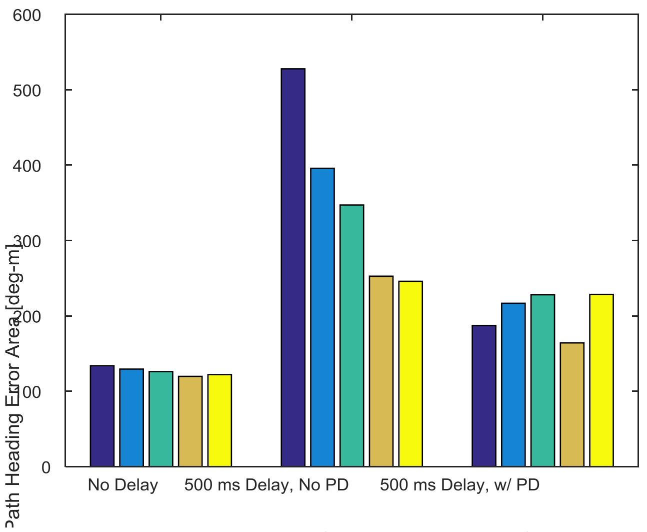

Figure 26. Integrated heading error for the 15 runs. First five represent configuration 1, second five represent configuration 2, and the third represents configuration 3.

$$\int_0^L |e_h(s)|ds \tag{29}$$

where $e_h(s)$ is the heading error. The value of these two metrics are plotted in Figures 25 and 26. For the configuration 2 it is observed that the trend is for the operator to drive more accurately from run to run. This is likely due to a training effect. These results are also shown in Table 3 where the relative improvement obtained by the predictive display over the case without the predictive display is demonstrated. The addition of the latency reduces the average speed from 46.5 kph to 32.5 kph (a 30% reduction) and the predictive display helps recover the speed lost due to latency indicating a 9% penalty in achieved speed. Regarding accuracy, it is observed in Table 3 that both metrics severely degrade with the addition of latency (148% for path deviation and 180% for heading deviation). For the predictive display the accuracy degrades, but not as severely. To gauge the benefit of the predictive display, its performance to the case without (configuration 2) is now compared. As shown in Table 3, the predictive display increases the overall speed by 29% and reduces the path deviation and heading error by 35% and 42% respectively. Finally, error is compared to the difference between the best and worst case as is done Zheng, et al. [15]. Using this normalized metric, the predictive display yields a 69% improvement in speed, a 59% improvement in path deviation and a 65% improvement in heading error.

This discussion concludes with some screen captures of an experimental run. Figures 27-30 show four screen captures on the course from configuration 3 (500 ms of delay with predictive display operational). Figure 27 shows operation on a straight section of the course. In this case the operator is driving on the ground plane and is moving toward the far plane. Although not entirely clear, the far plane occupies approximately 60% of the image and the ground plane occupies approximately 40% of the image. While in a mild turn, as illustrated in Figure 28, the predictive display still fills the whole image because the operator is driving into the scene which provides some margin of the far and ground plane to be displayed during mild turns (as illustrated in Figure 7). This is not the case for sharp turns such as the ‘S’ turn portion

Table 3. Results from experimental runs. The third and fourth column show performance and the percent difference in performance as compared to the best case (i.e. configuration 1). The fifth column represents the improvement realized by the predictive display for the 500 ms delay case (i.e. configuration 3 compared to configuration 2). The sixth and last column represents the normalized improvement as is done by Zheng, et al. in [24].

	Config. 1	Config. 2 (% difference to config. 1)	Config. 3 (% difference to config. 1)	Improvement of PD	Normalized Improvement of PD
Delay [ms]	0	500	500		
Predictive Display	N	N	Y		
Average Speed [kph]	46.5	32.5 (-30%)	42.2 (-9%)	29%	69%
Average Integrated Deviation [m^2]	1940	4816 (+148%)	3110 (+60%)	35%	59%
Average Integrated Heading Error [deg-m]	126	353 (+180%)	204 (+62%)	42%	65%

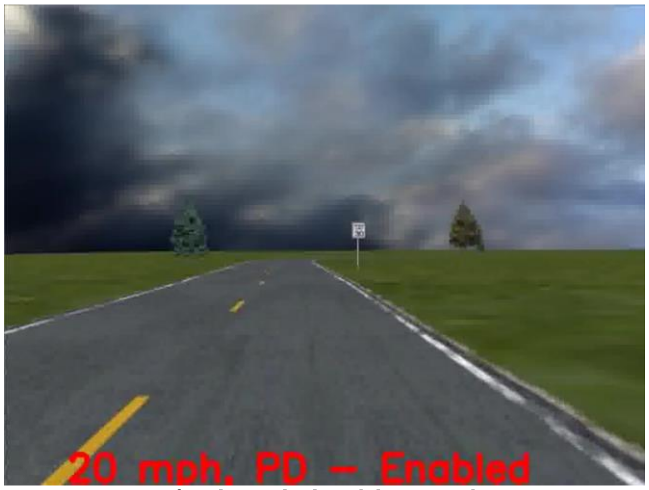

Figure 27. View of predictive display while on straight section.

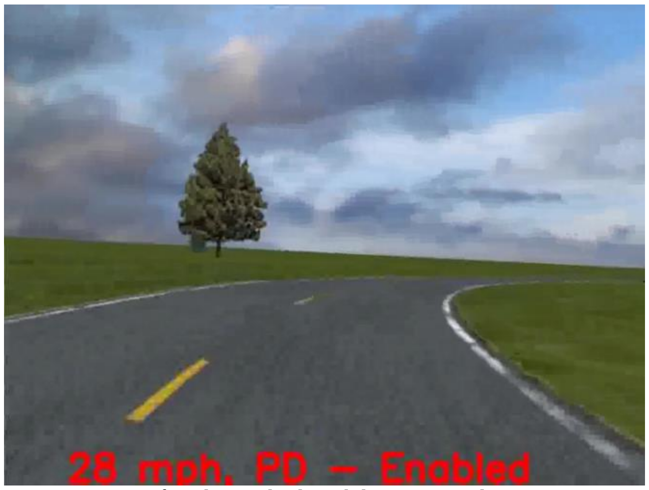

Figure 28. View of predictive display while on 75 m radius turn.

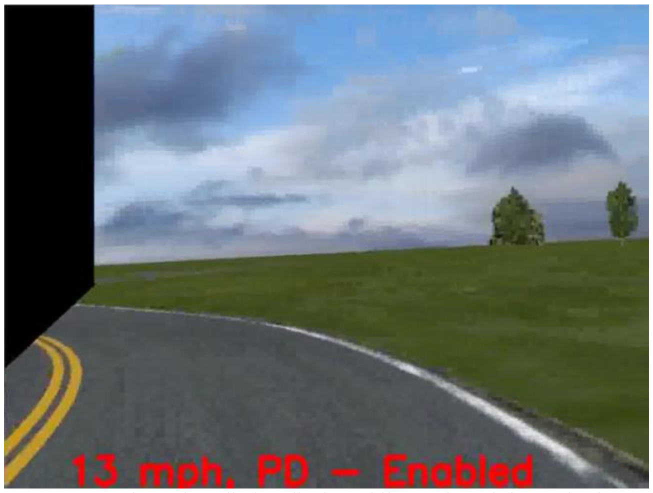

Figure 29. View of predictive display while on sharp left turn in the 'S' turn section (20 m radius).

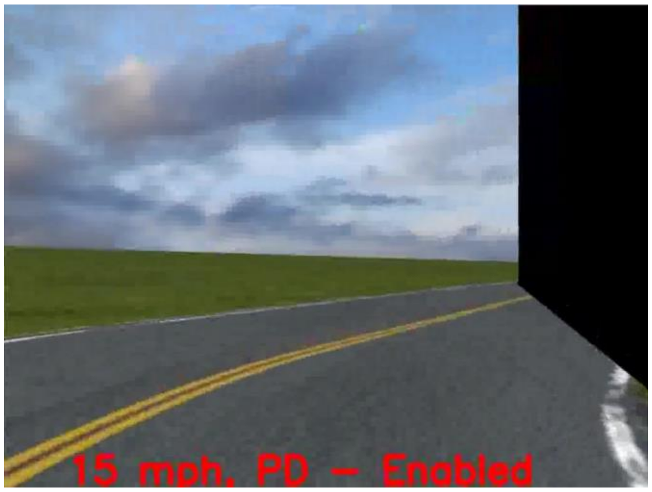

Figure 30. View of predictive display while on sharp right turn in the 'S' turn section (20 m radius).

of the terrain as illustrated in Figures 29 and 30. In these cases the manipulated image does not have content to represent the portion of the scene that it turned into. This will be mitigated in future by capturing a wider field of view than is presented to the operator.

CONCLUSIONS

In this paper a scheme for the mitigation of latency in the teleoperation of a UGV was presented. A state estimator was developed which has both feedforward and feedback functions to estimate the position of the vehicle over the round trip network delay. This information was used to manipulate the video frames being sent from the vehicle to the OCU to render a best estimate of what the operator would see in the no delay case. The implementation of this scheme in a simulation environment was then described. Preliminary experimental results were presented in which the predictive display was shown to be an effective method for the mitigation of latency by increasing achieved speed and by reducing the path deviation and the heading error significantly. By implementing predictive displays as a mitigation of latency in teleoperation, a minimally invasive approach to teleoperation was developed which has the potential for broad application to several UGV types and missions.

REFERENCES

[1] M. Brudnak, M. Pozolo, V. Paul, S. Mohammad, W. Smith, M. Compere, J. Goodell, D. Holtz, T. Mortsfield and A. Shvartsman, "Soldier/Hardware-in-the-loop Simulation-based Combat Vehicle Duty Cycle Measurement: Duty Cycle Experiment 2," in *SISO*

Simulation Interoperability Workshop, Spring, Orlando, FL, 2007.

[2] D. Cobzas and M. Jagersand, "Tracking and Predictive Display for a Remote Operated Robot using Uncalibrated Video," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation, ICRA 2005*, 2005.

[3] R. Held, A. Efstathiou and M. Breene, "Adaptation to displaced and delayed visual feedback from the hand," *J. Exp Psych*, vol. 72, pp. 871-891, 1966.

[4] M. Brudnak, M. Pozolo, A. Meldrum, T. Mortsfield, A. Shvartsman, W. Smith, J. Goodell and D. Holtz, "Virtual Combat Vehicle Experimentation for Duty Cycle Measurement," *SAE Int. J. Commer. Veh.*, vol. 1, no. 1, pp. 54-70, 2009.

[5] V. Paul, M. Brudnak, J. Ueda and A. Shvartsman, "Simulation-based Hybrid-Electric Combat Vehicle Duty Cycle Measurement," in *Intelligent Vehicle Systems Symposium, NDIA*, Traverse City, MI, 2006.

[6] M. Simon, M. Compere, T. Connolly, C. Lors, W. Smith and M. Brudnak, "Hybrid Electric Power and Energy Laboratory Hardware-in-the-Loop and Vehicle Model Implementation," in *SAE World Congress*, Detroit, MI, 2006.

[7] T. Ersal, R. B. Gillespie, M. Brudnak, J. L. Stein and H. K. Fathy, "Effect of coupling point selection on distortion in internet-distributed hardware-in-the-loop simulation," in *American Control Conference*, San Francisco, CA, 2011.

[8] T. Ersal, M. Brudnak, J. L. Stein and H. K. Fathy, "Statistical transparency analysis in internet-distributed hardware-in-the-loop simulation," *IEEE/ASME Transactions on Mechatronics*, vol. 17, no. 2, pp. 228-238, 2012.

[9] T. Ersal, M. Brudnak, A. Salvi, J. L. Stein, Z. Filipi and H. K. Fathy, "Development and model-based transparency analysis of an Internet-distributed hardware-in-the-loop simulation platform," *Mechatronics*, vol. 21, no. 1, pp. 22-29, 2011.

[10] T. Ersal, Y. Kim, A. Salvi, J. Siegel, A. Stefanopoulou, J. L. Stein, M. J. Brudnak and Z. Filipi, "A method to achieve high fidelity in internet-distributed hardware-in-the-loop simulation," in *NDIA Ground Vehicle Systems Engineering and Technology Symposium*, Troy, MI, 2011.

[11] T. Ersal, M. Brudnak, A. Salvi, Y. Kim, J. B. Siegel and J. L. Stein, "An Iterative Learning Control Approach to Improving Fidelity in Internet-Distributed Hardware-in-the-Loop Simulation," *Journal of Dynamic Systems, Measurement, and Control*, vol. 136, no. 6, 2014.

[12] X. Ge, M. Brudnak, J. L. Stein and T. Ersal, "A Norm Optimal Iterative Learning Control Framework towards Internet-Distributed Hardware-In-The-Loop Simulation," in *American Control Conference*, Portland, OR, 2014.

[13] A. Tandon, M. J. Brudnak, J. L. Stein and T. Ersal, "An observer based framework to improve fidelity in internet-distributed hardware-in-the-loop simulations," in *Dynamic Systems and Control Conference*, Palo Alto, CA, 2013.

[14] X. Ge, Y. Zheng, M. J. Brudnak, P. Jayakumar, J. L. Stein and T. Ersal, "Performance Analysis of a Model-Free Predictor for Delay Compensation in Networked Systems," in *IFAC Time Delay Systems Workshop*, Ann Arbor, MI, 2015.

[15] Y. Zheng, M. Brudnak, P. Jayakumar and T. Ersal, "An Experimental Evaluation of a Model-Free Predictor Framework in Teleoperated Vehicles," in *13th IFAC Workshop on Time Delay Systems*, Istanbul, 2016.

[16] X. Ge, M. Brudnak, P. Jayakumar, J. L. Stein and T. Ersal, "A Model-Free Predictor Framework for Tele-Operated Vehicles," in *American Control Conference*, Chicago, IL, 2015.

[17] F. Chucholowski, S. Buchner, J. Reicheneder and M. Lienkamp, "Prediction Methods for Teleoperated Road Vehicles," in *Conference on Future Automotive Technology - Focus Electromobility*, Munich, Germany, 2012.

[18] F. Chucholowski, M. Sauer and M. Lienkamp, "Evaluation of Display Methods for Teleoperation of Road Vehicles," in *9th International Conference on Intelligent Unmanned Systems*, 2013.

[19] D. Lovi, N. Birkbeck, A. H. Herdocia, A. Rachmielowski, M. Jagersand and D. Cobzas, "Predictive Display for Mobile Manipulators in Unknown Environments Using Online Vision-based Monocular Modeling and Localization," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[20] A. Rachmielowski, N. Birkbeck and M. Jägersand, "Performance evaluation of monocular predictive display," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, Anchorage, AK, 2010.

[21] E. Royer, M. Lhuillier, M. Dhome and J.-M. Lavest, "Monocular Vision for Mobile Robot Localization and Autonomous Navigation," *International Journal of Computer Vision* , vol. 74, no. 3, p. 237–260, 2007.

[22] A. Kelly, N. Chan, H. Herman, D. F. Huber, R. Meyers, P. Rander, R. Warner, J. Ziglar and E. capstick, "Real-

Time Photorealistic Virtualized Reality Interface for Remote Mobile Robot Control," in *Proceeding of International Symposium of Robotics Research*, 2009.

[23] R. Szeliski, Computer Vision: Algorithms and Applications, New York: Springer, 2010.

[24] Itseez, Inc., "OpenCV," [Online]. Available: http://opencv.org/. [Accessed 23 May 2016].

[25] Realtime Technologies, Inc, "SimCreator," Realtime Technologies, Inc., 2016. [Online]. Available: http://simcreator.com/simcreator/simcreator.htm. [Accessed 6 May 2016].

[26] R. Romano, "Realtime Driving Simulation Using A Modular Modeling Methodology," in *SAE World Congress*, Detroit, MI, 2000.

[27] R. Romano, "Real-Time Multi-Body Vehicle Dynamics Using a Modular Modeling Methodology," in *SAE World Congress*, Detroit, MI, 2003.

[28] B. Dawes, D. Abrahams and R. Rivera, "Boost C++ Libraries," May 2016. [Online]. Available: http://www.boost.org/. [Accessed 6 May 2016].

[29] F. Bellard, "FFmpeg," 2016. [Online]. Available: http://ffmpeg.org/. [Accessed 10 May 2016].