2016 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
AUTONOMOUS GROUND SYSTEMS (AGS) TECHNICAL SESSION
AUGUST 2-4, 2016 – NOVI, MICHIGAN

BLAST: BANDWIDTH AND LATENCY-SCALABLE TELEOPERATION[1]

Chris L Baker, PhD,
Parag Batavia, PhD
Neya Systems
Wexford, PA[2]

ABSTRACT

This paper describes an approach to aid the many military unmanned ground vehicles which are still teleoperated using a wireless Operator Control Unit (OCU). Our approach provides reliable control over long-distance, highly-latent, low-bandwidth communication links. The innovation in our approach allows refinement of the vehicle's planned trajectory at any point in time along the path. Our approach uses hand-gestures to provide intuitive fast path editing options, avoiding traditional keyboard/mouse inputs which can be cumbersome for this application. Our local reactive planner is used for vehicle safeguarding. Using this approach, we have performed successful teleoperation nearly 1500 miles away over a cellular-based communications channel. We also discuss results from our user-tests which have evaluated our innovative controller approach with more traditional teleoperation over highly-latent communication links.

INTRODUCTION

Fully autonomous ground vehicle technology has been developing rapidly over the past several years, but is not yet dependable enough for most real-world defense applications operating in unstructured environments. The vast majority of military unmanned ground vehicles are teleoperated with a human operator directing the vehicle's motions through a wireless Operator Control Unit (OCU). This control methodology relies either on a direct line of sight with the vehicle or on relayed video feedback. Therefore most teleoperated ground vehicle systems require a high-bandwidth low-latency link to support either real-time streaming video or streamed 3D structure back to the OCU. This approach experiences significant performance degradation in the presence of communication latencies as small as 100 milliseconds. These latency and bandwidth requirements severely hamper the use of teleoperated systems in situations where latency is high and/or bandwidth is low.

We have demonstrated effective control of a remote vehicle in the presence of communication latencies of 500 milliseconds to 5 seconds in simulation as well as controlling

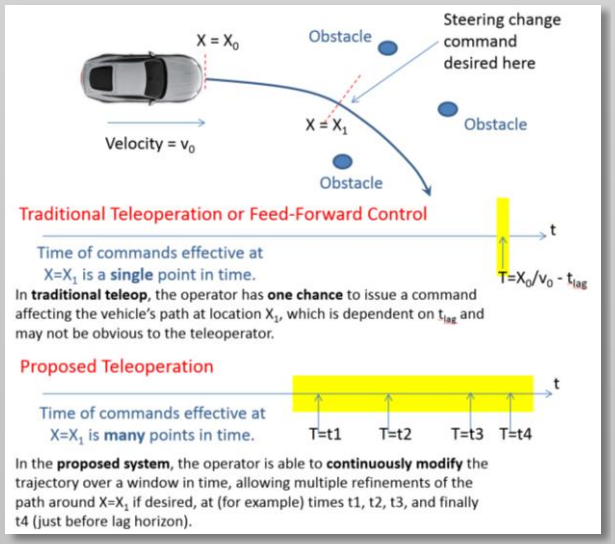

Figure 1 – In Contrast to traditional teleop (middle), BLAST teleoperation provides the operator with multiple chances to refine the vehicle's trajectory at any location (bottom).

[1] This material and research was based upon work supported by the United States Army under Contract No. W56HZV-14-C-0125 with Dr. Gregory Hudas at TARDEC.

[2] Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the United States Army.

Figure 2 – Top-down user-perspective view of realistic animated hand laying out a path for the vehicle to follow.

an actual test bed vehicle over a cellular network nearly 1500 miles away. We have achieved this goal by combining a vehicle safeguarding approach based on Neya's Receding-Horizon Model-Predictive (RHMP) controller with a novel approach for high-dimensional control of the vehicle's immediate path. The ability for the remote teleoperator to shape and refine any part of the vehicle's future trajectory is a key benefit to our approach. As Figure 1 illustrates, traditional teleop control can only modify the path at a single specific time in front of the vehicle which is dependent on the current communication latency. Our approach mitigates the latency problem by enabling multi-point control over the entire planned path in front of the vehicle.

This trajectory control is only helpful when the user has an effective means for rapidly specifying and modifying the trajectory in real time. In our system, the user interacts with the trajectory using various gesture-based controller devices. Natural motion input through the use of gesture-recognition offers a comfortable way of performing remote vehicle control. These input devices have been demonstrated to enable complex maneuvers naturally while reducing the complexity of the interface where the control is enacted [1], [2]. Additionally, the visual feedback we provide in the 3D user interface enables fast and intuitive learning of the control interface by mirroring the user's own form in the display [3] such as that displayed in Figure 2.

The user can make use of the gesture-based input to specify a global path to the vehicle by adjusting a path's spline control points directly. These control points are used to generate a smooth path for the vehicle to traverse. By allowing the user to modify any point of the path at any future point in time, we are distinctly different from other teleop approaches which only allow editing the path at a single point in time.

Several approaches have been made to improve the teleoperation experience, summarized well in [4] by providing limited local context, situational awareness, and adapting to communication latencies. Their suggestions however, and indeed all the approaches we have found, apply vehicle control to a single point in time. Other research has demonstrated that traditional joystick and steering wheel type control is non-optimal when controlling a vehicle remotely [5] [6], and ongoing studies have suggested other input interfaces such as gesture inputs may be superior to these traditional approaches [7].

Recent work on the DARPA-funded AVATAR program has demonstrated successful control of a vehicle over a highly-latent transcontinental satellite communications link [8]. Their approach works by driving a virtual vehicle in front of the live vehicle. While this approach enables traditional vehicle control over highly latent communications, this approach also suffers from providing the user only one chance to issue vehicle control at any instant in time; that control cannot later be modified.

Our approach provides a corrective opportunity by allowing the user to adjust any part of the path in front of the vehicle that has not yet been consumed by the latent communications. As the vehicle moves, the local environment may change due to new terrain information from the robot's perception system. When this happens, the user may need to change the vehicle's planned trajectory path and has two options for doing this. They can adjust the vehicle's trajectory using our path modification tools as long as the path has not been consumed by the latency indictor, or they can allow the desired path to remain and rely on the reactive planner to safe guard the vehicle by automatically planning trajectories around obstacles. In highly latent situations this control modality provides the user with a fine level of path control while still safe-guarding the system and allowing the user to modify and refine the path prior to execution.

Dynamic Path-Based Control Generation

BLAST uses a control mechanism which provides the user with an opportunity to send paths to the vehicle through a dynamically changing 3D world. The interface we have developed enables the user to lay down paths and edit them quickly by adjusting spline control points primarily using hand gestures. Because the direct vehicle control is latent, we also require a local vehicle safeguarding approach.

Gesture-Based Controller Input

Our current implementation optionally uses one of several possible controller inputs: a standard joystick controller, a Wii-mote controller, and a Leap Motion Controller™ [9]. In this paper we focus primarily on the Leap interface. We use the Leap to detect the hand and finger positions of the user. Based on these positions, we allow for path generation and enable several path modification tools. When creating paths for the vehicle to drive, the thumb, index finger, and pinky are used to control three spline points. Starting at the last point on the path and tangent to the last portion of the path, the three spline points are used to create a smooth curve for the vehicle

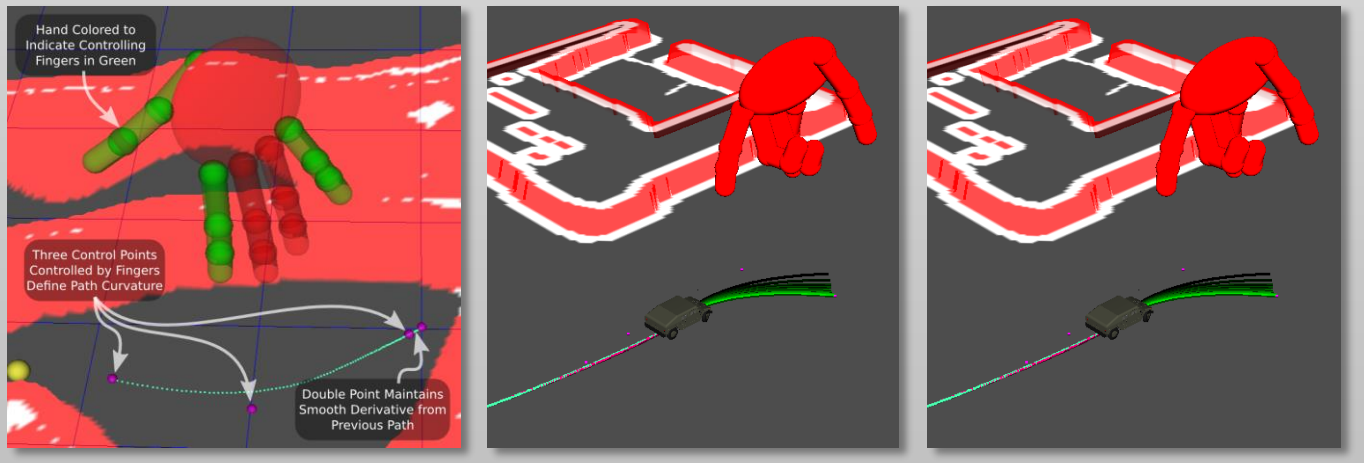

Figure 3 – The ground-intersection of the finger directions are used to add splines to the end of the path.

to follow. The spline points are found by extending the three finger directions and intersecting with the ground plane (Figure 3).

Once the user is satisfied with the layout of the path, a key on the keyboard is pressed to commit the path. This path is sent to the vehicle and the vehicle immediately begins driving the provided path. At this point, the user's perspective is moved to the end of the path, where a new set of control points allow for generation of path extensions. This process continues as the user lays down new paths to be followed until the goal is reached.

Because the user is capable of laying paths down outside the sensor's range, and because the surrounding world is dynamic, updated data providing the latest local context is transmitted back to the user control station. This data currently is in the form of a costmap, showing where the drivable and un-drivable areas of the map are located. Because the surrounding area is dynamic, and because the control of the vehicle is highly latent, it becomes extremely critical to have a local reactive planner running on the vehicle to provide vehicle safeguarding.

Local Reactive Planning

We use Neya's locally running safeguarding autonomy system which incorporates a local reactive planner called AM3P: Adaptive Modular Multi-Terrain Mobility Planner developed primarily with SPAWAR on the EV1 HMMWV ONR Code 30 program. The vehicle uses a single stereo camera system to generate costmaps of an area in front of the vehicle. This costmap is used both by the local reactive planner as well as sent back to the user for vehicle control.

A key advantage of AM3P is that it operates directly in control space, allowing for accurate modeling of vehicle dynamic limits including control lag, steering-wheel rate change, and acceleration/braking constraints. AM3P also exploits a library of sophisticated pre-computed maneuvers as needed via selective invocation of a combinatorial motion lattice planning module providing real-time use even during very complex motions.

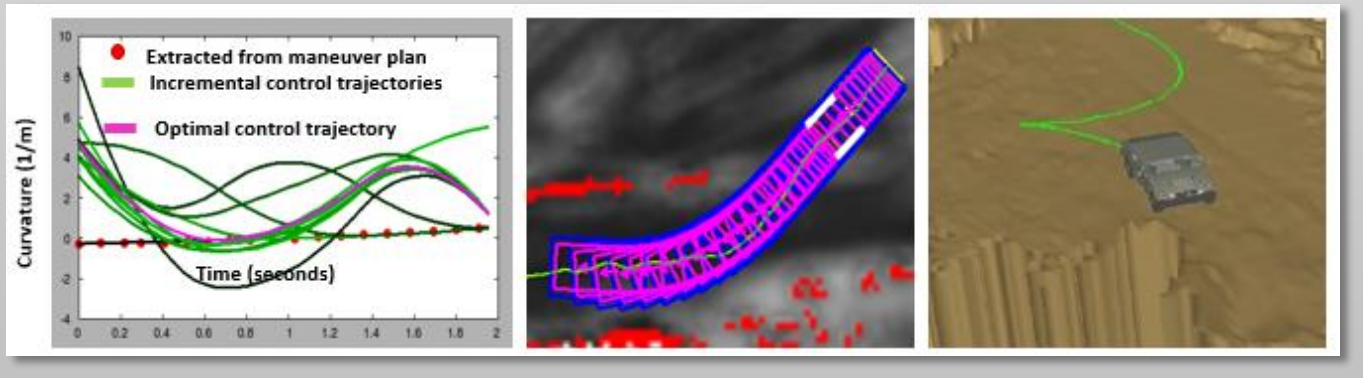

Figure 4 – Neya's approach selects the optimal control input by forward simulating trajectories over a five-second time horizon.

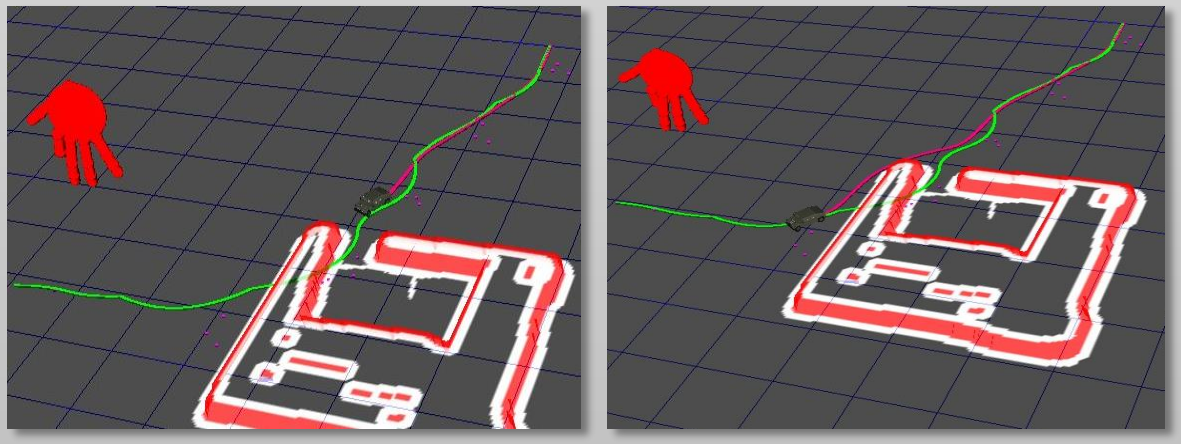

Figure 5 – Planner Safeguarding: User lays a path which intersects an obstacle (left). The global planner plans a path to adhere as closely as possible to the user's desired path, but avoids going through the obstacle (right).

The existing system employs selective reasoning about individual wheel placement, factoring in vehicle underbody clearance, roll, and pitch, to achieve accurate navigation in off-road terrain without risk of damaging the chassis or suspension.

Our variant of model-based receding horizon model predictive control optimizes the control values (velocity and steering commands) sent down to the low-level vehicle controller by evaluating candidate simulated trajectories of the vehicle forward in time with respect to the desired path. Both the control space and cost metric for the RHMP planner are treated as continuous functions, using numerical optimization approaches to calculate control sequences which minimize overall trajectory cost. By explicitly optimizing over vehicle control, the system will accurately plan trajectories which follow the desired path while respecting dynamic and kinematic limitations of the platform (Figure 4). This approach to the reactive planning and vehicle safeguarding works well for the BLAST system, providing the safeguarding capability needed for remote latent teleop.

Local Vehicle Safeguarding

Using the AM3P reactive planner we are able to respond to local environment changes that may need to be avoided before the user can enact new controller paths around the obstacle. Figure 5 demonstrates this local safe guarding in action. The current path to the vehicle (left) is passing through an obstacle, but the reactive planner responds correctly and avoids the obstacle while also staying as close as possible to the user's desired path.

The amount of deviation allowed by AM3P is configurable. In some instances, the user may want the vehicle to solve local problems when possible. However, it is often the case in teleoperation mode that the user will want the vehicle to adhere as close as possible to the desired path. In these instances, if the vehicle encounters a blocked path, the vehicle will stop forward progress (safely) providing the user with time to adjust the desired path even when operating within a highly latent communications connection. It is this real-time adjustment of the path capability that sets the BLAST approach from other teleoperation approaches. Once the user has adjusted the path using the BLAST path adjustment tools, the vehicle will continue the traversal.

Dynamic Path Editing

Once a path has been sent to the vehicle, the dynamic nature of the local contextual data near the vehicle may prove to be un-traversable by the reactive planning system. For example, when using an *a-prior* map, new information may be available, or moving objects may result in an impassible

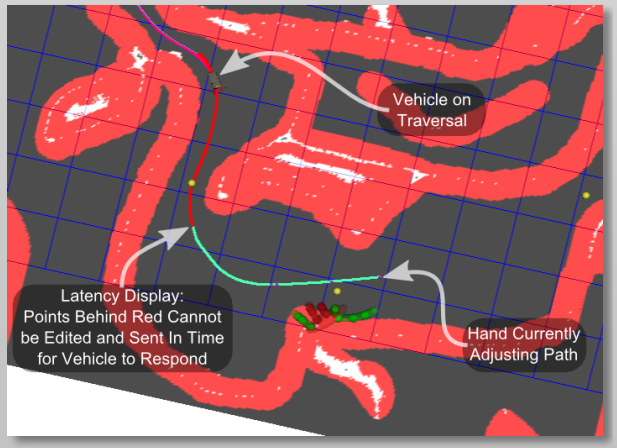

Figure 6 - The path changes to red in front of the vehicle indicating that any edits will not have time for the vehicle to respond due to the current latency estimate.

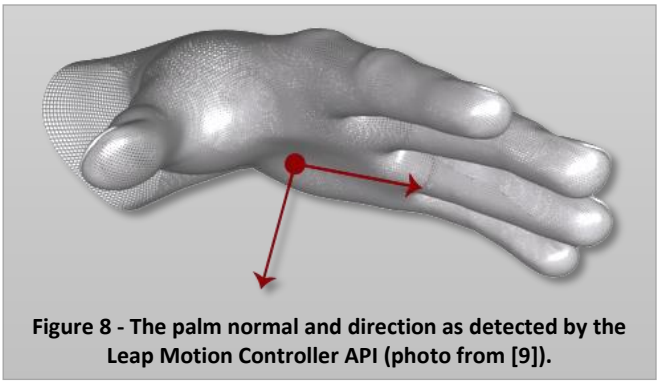

Figure 8 - The palm normal and direction as detected by the Leap Motion Controller API (photo from [9]).

region of the map. In these cases, the user may elect to refine the path currently being pursued by the vehicle. In traditional teleoperation, this is not possible. However, because our system uses a path-based spline control, the path can be easily edited providing the vehicle has not traversed path the latency point.

The latency point is the position along the path in front of the vehicle which can no longer be edited because any edits to that portion of the path will not arrive at the vehicle in time for the vehicle to make the necessary adjustments to follow any new path. Figure 6 illustrates this as the "Latency Display" point. As shown, the current editing point is well in front of the vehicle, but based on the vehicle's speed, and the estimated latency, the path will not be editable prior to the transition from red to green path.

Leap-Based Path Editing

Using the interface to the Leap Motion Controller, we can enable various editing modes. The editing modes we describe here are a sample of the possible modes and have proven to be quite useful when attempting to quickly edit paths to improve traversability. As the user is laying out a path, there may be reasons to adjust the previously committed path. This can happen, for example, if the user has laid a path in error, or if the perception data updates and presents a different environment, or possibly moving objects are present and need to be planned around. The user can navigate forward and backward along the path using the left and right arrow keys to select the position to apply the path edits. The user can also strike the up and down arrows to switch between three distinct editing modes.

In order to enforce connectivity to adjacent points while adjusting the path, we allow the user to adjust the control points of a resampled spline. The resampling is helpful because the original control points laid down may not be evenly spaced, so further adjustment would be difficult.

Each of the morphing modes provides a tool that the user can use to adjust the path. The user's palm is tracked in the Leap's workspace and the plane and direction of the palm are used to enact the tool's modifications to the path. Figure 8 shows the palm extraction provided by the Leap Motion API, and the palm's direction as extracted by the Leap Motion software. To help stabilize the extracted control vectors, we run a low-pass filter on the Leap's raw output.

The path's resampled control points are connected and modeled as a mass/spring system. The mass/spring system enables intuitive movement of the path as if it were a rubber-band connected set of points. The effect of the user's movements on any one control point is scaled by one over the square of the distance from the point being modified so that only portions local to the workspace are modified when the user is adjusting points.

The palm's extracted plane normal is intersected with the ground plane which defines the line for the tool. The tool is initially inactive and drawn in a light gray color to indicate the tool's location prior to activation. Holding down a key on the keyboard activates the tool. At this point, the user interacts directly with any control points captured within the circle of the tool's influence, depending on the size or scale of the tool. Once path modification is complete, the key is released and the active state of the tool is changed to inactive. Alternatively

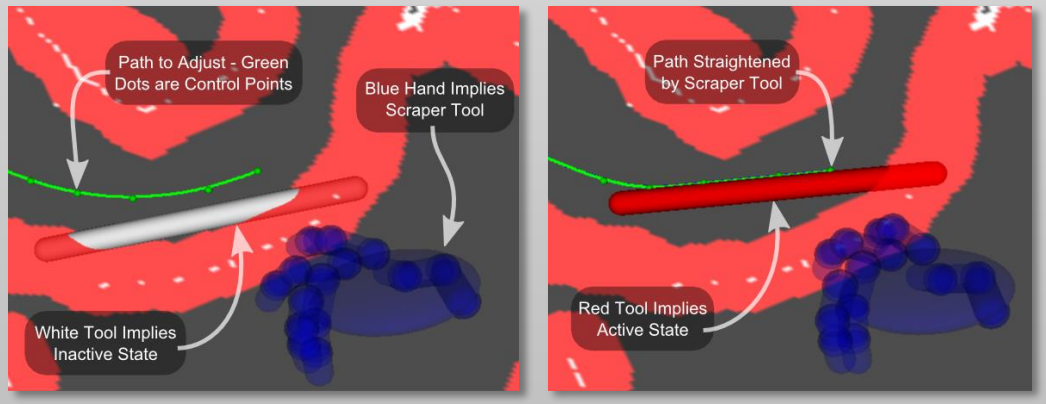

Figure 7 - The "Scraper" tool adjusts the path using a long flat tool and sliding the points.

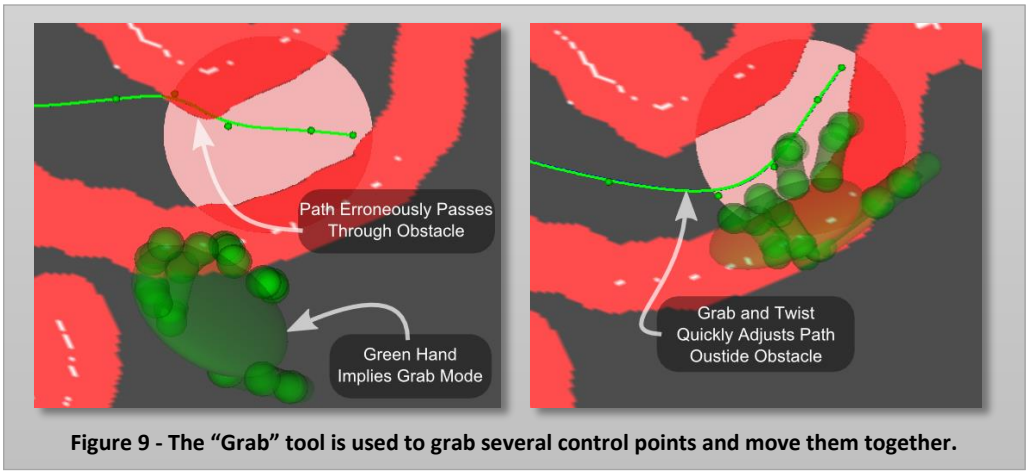

Figure 9 - The "Grab" tool is used to grab several control points and move them together.

the tool is deactivated by moving forward or back along the path using the arrow keys, or switching to another tool.

The user can also adjust the size, or scale of the tool by bending or extending the fingers. The palm is most reliably tracked by the Leap Motion Controller when the fingers are all extended, so this is when the tool is the smallest because it provides the best fine-detail control. As the number of extended fingers decreases, the size of the tool increases. This enables the user to dynamically and intuitively select the number of points on the path that should be adjusted.

Scraper Mode

The *"Scraper"* tool (akin to an ice-scraper) can be used to "scrape" or "slide" large sections of the path to adjust the control point's positions in a straight line. Figure 7 shows this tool in use. The tool is drawn on the ground in light gray until it is activated. At this point, the tool changes to a red color to indicate the active state. The rendered hand in the scene is changed to blue to further indicate the tool being used is the *"Scraper"* tool.

The scraper tool is best used to slide a selection of points in a straight line. This is good when the path needs to be smoothed out in long straight sections for more optimal traversals. The tool can also be used by placing it between two control points and performing a twisting motion. This will result in the vehicle following a slalom motion between small discrete obstacles.

Grab Mode

The *"Grab"* mode places a circle whose center is at the point of intersection with the palm's direction (a feature defined by the Leap Motion Controller API as in Figure 8) and the ground plane. This circle indicates which spline control points will be modified when the tool is activated. As the tool is activated, all control points within the tool's scope become fixed relative to the moving reference frame defined by the circle. Thus moving the user's hand, which changes the position of the palm's intersection with the ground plane, will change the position of all the control points while keeping their relative position constant. The rotation of the palm around the palm's normal direction is also tracked by the Leap Motion Controller, and thus the points can not only be translated as a group, but also rotated when the user's hand rotates.

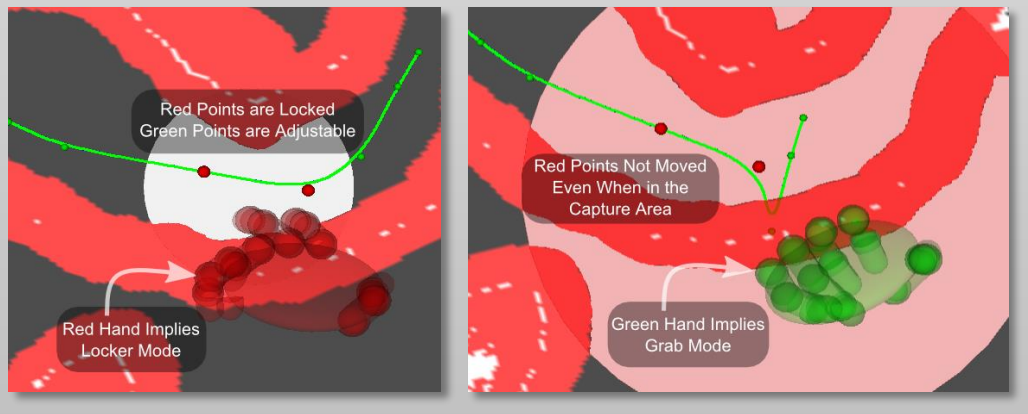

Figure 10 - The "Locker" tool locks points that the user does not want to be moved.

It is important to understand that in this mode, the captured points do not change their position relative to one another, but stay fixed relative to one another and change relative to the global path and global coordinates. This provides the user with a method to move large sections of the path while maintaining the path's local curvature.

Figure 9 shows the grab tool in use. Here the user is adjusting a poorly laid path which passes through an obstacle. As the user's hand rotates, the control points are adjusted such that the path no longer passes through an obstacle.

Locker Mode

The final mode that we have implemented is the *"Locker"* mode. In this mode, the same circle is drawn relative to the user's palm intersection with the ground as described previously in the *"Grab"* mode. The application of this tool toggles an editable state of any points within the tool's scope, thus locking or unlocking the points for morphing with the other tools. This provides the user with the critical ability to lock down points so they will not be editable either by the mass/spring system modeling for the control points, or by any of the selected tools. This is useful for example when the user perfectly places a set of control points through a tight corridor, but would like to adjust points nearby without editing the points in the corridor. Normally the mass/spring system may affect the points previously placed as described above. However, if the points are locked, the nearby path can be adjusted as needed, and the locked points will remain in place.

Figure 10 demonstrates this approach. In the left image, the user has selected and locked down two control points, indicated by the point's red color. The right images shows the user adjusting the path with a large scaled grab tool. The three green control points are captured and moved and the green path responds. However, the two locked red control points are stationary and maintain their position.

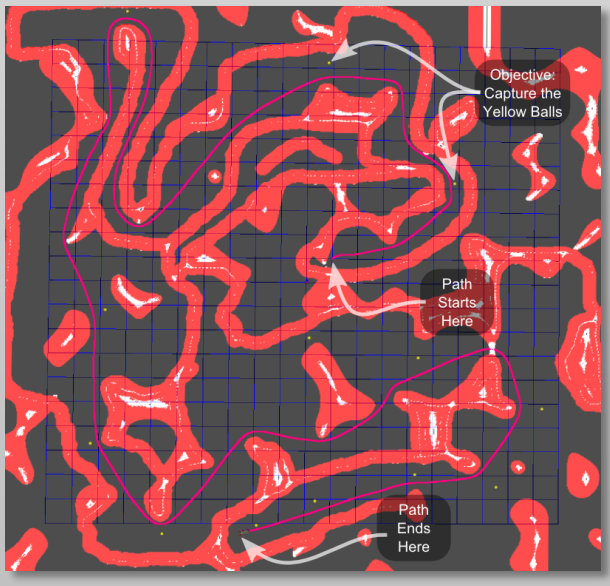

Figure 12 - The full course used for the user tests.

Simulation Validation

In order to validate the usefulness of our approach we have developed a simulation to perform some limited user testing and validation.

Experiment Description

We asked users to navigate a simulated vehicle through the obstacle course shown in Figure 12. This test exercised various control methods and allowed us to compare several user's performance against a more traditional joystick-based teleop approach with varying amount of latency.

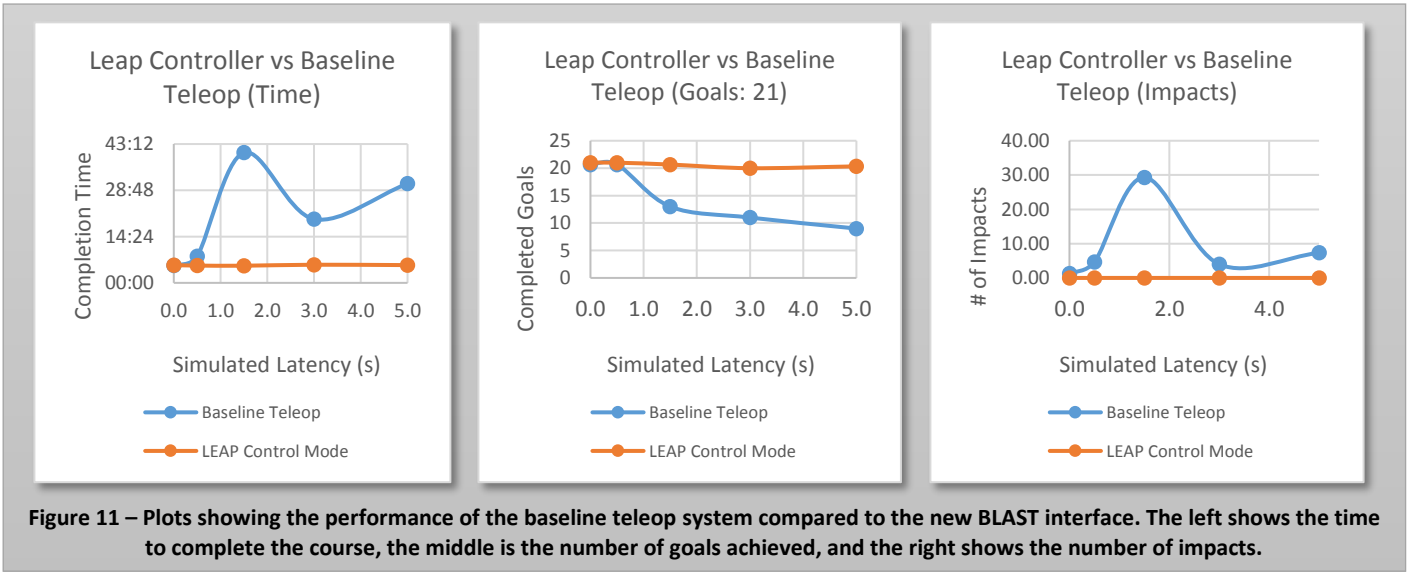

Figure 11 – Plots showing the performance of the baseline teleop system compared to the new BLAST interface. The left shows the time to complete the course, the middle is the number of goals achieved, and the right shows the number of impacts.

For all tests, the vehicle was started in the same place in the middle of the course. The objective of the test was to capture all the yellow balls by driving the vehicle over them in as short a time as possible. The test was run at several different latencies.

If the system detected a vehicle/obstacle impact, the display alerted the user by flashing a message, "Warning – Impact Detected" in red in the center of the screen. When this happened, the user was only able to extract the vehicle from the obstacle by moving in reverse. While the vehicle was detected to be in an obstacle, the timer was sped up 10x to provide adequate incentive for avoiding obstacles. The number of impacts was also tracked and recorded for each run.

The users evaluated the system against a baseline approach. The baseline approach we implemented used a standard joystick control mechanism. As the latency increases, this method of control becomes unusable. Several groups have cited the difficulty in latencies approaching the 1-2 second range [10], [11], and [12]. Our own user tests also demonstrate this as well. We demonstrate that latencies as small as 1.5s make the joystick control method frustrating to the user and completely unusable.

To make the test more real, and avoid the user simply "learning" the latency, we also added a random walk component to the amount of latency being added in the system. The random walk was bounded, but was allowed to vary by $\pm 0.5s$.

User Test Results

Figure 11 summarizes the test results. The left shows the time required to complete the course. The user's time for the Leap-based system was very close to the absolute minimum time required from an autonomously planned minimum distance run which did not consider capturing the goals. The middle plot shows the number of goals achieved. This metric is impacted by the fact that many users found the baseline system so unbearable with more than 3.0s latency that they gave up or didn't reverse to pick up missing goals. Finally, the right plot shows the number of impacts from each approach. Notice that when using the BLAST system there were zero impacts because the autonomy system actively and successfully avoided impacting any obstacles.

Observations

During the user tests, we noticed the amount of time for the user to become familiar with the BLAST control strategies was surprisingly short. A few minutes of training and orientation with the system resulted in reliable control. Further improvements to the user's time during subsequent tests was mostly spent optimizing the path by entering one of the morph modes and moving and adjusting the points prior to being overtaken by the vehicle's estimated latency.

Notable in our results is the fact that the performance for users using the new BLAST system was not affected at all by the additional latency. We believe that a 5s latency is not the upper bound to how the vehicle can be controlled, given enough range in the perception data.

We also noticed a curious improvement in the baseline performance from the 1.5s latency to the 3.0s latency. After interviewing the test subjects we found that the most likely explanation for this phenomenon was a distinct change in control approach once the latency grew beyond a certain amount. With low levels of latency the users would try and control the vehicle in the normal way, just predicting the latency while driving with reasonable success. The subtle drop in performance was not significant enough to motivate a change in their control approach. Once the latency grew too high for continual uninterrupted control, the users would change their control approach by sending commands to the vehicle in very short spurts and wait to see the vehicle's response (this is similar to control strategies used for extreme long range teleoperation and control). Counting or mental timing helped to extend the length of the control spurt, but the painful process of waiting for the response from the vehicle before continuing to drive was inevitable.

Some of the users found the vehicle so completely uncontrollable after only 3.0s of latency (using the baseline system without BLAST) that they were unable to complete the task. This result further encourages the use of our more advanced control strategies to help control the vehicle during times of high latency.

Live Vehicle Testing Results

To further validate our control approach, we also have performed live vehicle testing on Neya's UxInterceptor platform shown in Figure 13. We accomplished remote control of the vehicle in two distinct configurations. The first

Figure 13 – Neya Systems autonomy testbed.

tests were performed while the vehicle was connected to the Neya Systems intranet via a EZBR-0214 wireless bridge. The bridge acts as a single dedicated wireless link between any two Ethernet ports. The link has a 3 mile range, though we saw degraded performance for anything over about 0.5 miles. These units are also highly directional and so using them for testing enabled analysis over a weak communication link in preparation for the more aggressive cellular based configuration.

The second test was performed using a communications link consisting of a Verizon Mifi unit connected to a 4G cellular network. All network traffic with the vehicle used UDP communications while connected to a Virtual Private Network (VPN) to join the various nodes on the network. We have tested this configuration of vehicle control from as little as a 5 mile separation to as much as a nearly 1500 mile separation between the vehicle and the controller, with the controller station located near Denver Co and the vehicle located in Pittsburgh Pa.

Future Work

While control using the low-bandwidth costmap-only data is possible and will allow for safe vehicle control from long distances over highly latent communication links, we are currently working to improve this technology to provide a much higher fidelity source of data to improve the situational awareness and contextual information such that the operator can more powerfully control the vehicle in challenging environments.

When the vehicle arrives at an impassible element in the costmap, the user is often required to re-plan a path around the obstacle to continue forward progression. However, today's perception systems often provide false alarms, indicating obstacles where there are none. We are currently working on an approach that will allow the user to not only edit the path's provided to the vehicle, but also adjust the costmap to inform the perception and local planning and safeguarding systems that the obstacle is indeed passable.

In addition to the Leap controller interface, we are evaluating other controller modalities and data representation to improve the situational awareness and local contextual information provided to the user to enable a higher-fidelity of remote control.

References

[1] A. Uribe, S. Alves, J. M. Rosario, B. Perez-Gutierrez and others, "Mobile robotic teleoperation using gesture-based human interfaces," in *Robotics Symposium, 2011 IEEE IX Latin American and IEEE Colombian Conference on Automatic Control and Industry Applications (LARC)*, 2011.

[2] C. Guo and E. Sharlin, "Exploring the Use of Tangible User Interfaces for Human-robot Interaction: A Comparative Study," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 2008.

[3] O. Portillo-Rodriguez, O. O. Sandoval-Gonzalez, C. Avizzano, E. Ruffaldi and M. Bergamasco, "Capturing and Training Motor Skills," in *Human-Robot Interaction*, 2010.

[4] J. Y. Chen, E. C. Haas and M. J. Barnes, "Human performance issues and user interface design for teleoperated robots," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 37, no. 6, pp. 1231-1245, 2007.

[5] T. Fong, C. Thorpe and C. Baur, "Advanced interfaces for vehicle teleoperation: Collaborative control, sensor fusion displays, and remote driving tools," *Autonomous Robots*, vol. 11, no. 1, pp. 77-85, 2001.

[6] T. W. Fong, F. Conti, S. Grange and C. Baur, "Novel interfaces for remote driving: gesture, haptic, and PDA," in *Intelligent Systems and Smart Manufacturing*, 2001.

[7] T. H. Song, J. H. Park, S. M. Chung, S. H. Hong, K. H. Kwon, S. Lee and J. W. Jeon, "A Study on Usability of Human-robot Interaction Using a Mobile Computer and a Human Interface Device," in *Proceedings of the 9th International Conference on Human Computer Interaction with Mobile Devices and Services*, New York, NY, USA, 2007.

[8] A. Kelly and P. Rander, "AVATAR," 10 2014. [Online]. Available: www.nrec.ri.cmu.edu/about/news/14_10_avatar_teleoperation.php.

[9] *Leap Motion, url: https://www.leapmotion.com/.*

[10] Witus, Gary, Shawn Hunt, and Phil Janicki. "Methods for UGV Teleoperation with High Latency Communications," *SPIE*, May 2011.

[11] Cossell, S., M. Whitty, and J. Guivant. "Streaming Kinect data for robot teleoperation," *Proceedings of the 2011 Australasian Conference on Robotics and Automation*, 2011.

[12] Lou, Ren C., Tse Min Chen. "Development of a multi-behavior based mobile robot for remote supervisory control through the internet," *IEEE/ASME Transactions on Mechatronics*, pp. 5(4):376-385, December 2000.