A holistic framework for planning, real-time control and model learning for high-speed ground vehicle navigation over rough 3D terrain.

Nima Keivan[1], Steven Lovegrove[1] and Gabe Sibley[1]

***Abstract*— This paper describes a local planning, control and learning framework enabling high-speed autonomous ground-vehicle traversal of rough 3D terrain replete with bumps, berms, banked-turns and even jumps. We propose an approach based on fast physical simulation and prediction, which we find offers numerous benefits: first, it takes advantage of the full expressiveness of the inherently non-linear, highly dynamic systems involved; second, it allows for the fusion of local planning and model-based feedback control all within a single framework; third, it allows vehicle model learning. The final and most important reason to use physical simulation as a unifying framework is that it works well in practice. The system is experimentally validated on a high speed nonholonomic remotely controlled vehicle on undulating terrain using a scanned 3D ground model and motion capture ground-truth data. Parameter reduction is achieved with the use of cubic curvature control primitives and a fast precomputed lookup table.**

I. INTRODUCTION

Recent developments in path planning and navigation have enabled operation in increasingly challenging environments. The use of motion primitives [9] and stochastic search methods such as RRT and RRT* [8] [6] have resulting in algorithms that successfully navigate complex obstacle fields even in higher order configuration space. A major advantage of these methods is that they can employ nonlinear dynamics models thereby enabling physically accurate planning in complex environments without approximation or linearization. However, this advantage comes at a performance price as stochastic methods invariably sample infeasible trajectories. Conversely, optimization based methods [4] employ effective initial guesses and numerical or analytical optimization techniques to rapidly converge on optimal paths. However due to the reliance on the accuracy of the initial guess, these methods are susceptible to failure or suboptimal performance depending on the quality of this guess.

The quality, optimality and methodology of the plans notwithstanding, their open loop performance in real robots is inevitably impaired by the existence of imperfections or extraneous inputs that may not have been included in the dynamics model. Therefore for real-life applications, some form of closed loop control is desired. Moreover, both the planner and control systems rely on an accurate model in order to properly control and plan for the robot. Due to the difficulty of obtaining accurate model parameters, it is desirable to learn model parameters by observing the response of the robot to control inputs. Recent developments in Model Predictive Control(MPC) [3] and Learning-based Model Predictive Control (LBMPC) [2] [10] have strived to both implement model-based control schemes and to improve the underlying model parameters by observing the response of the system to inputs. The advantage that these schemes hold over more traditional control methods is twofold: the incorporation of increasingly complex models, and the ability to generate control policies over a predicted series of timesteps into the future. The latter offers clear advantages when controlling an infeasible trajectory or one that was made using an inaccurate model.

An alternative to model predictive control, traditional feedback systems use static and/or dynamic feedback of the state to determine the controls for the next time steps. Recent developments in this field have resulted in methods allowing the calculation of Lyapunov functions for nonlinear systems [5] and defining graphs of Lyapunov-stable region around states as in the case of LQR-Trees [11]. Generally speaking, these methods rely on the linearization of the state transfer function in order to analytically obtain control policies.

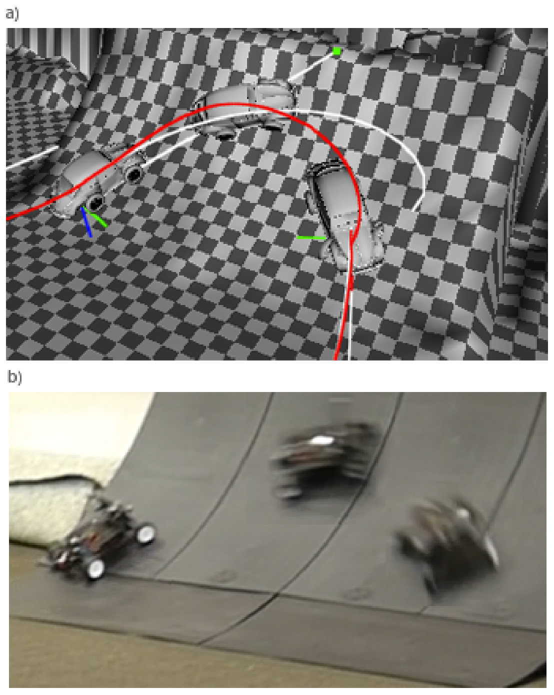

Fig. 1. a) Local plan (red curve) generated between two points on a 3D scanned quarter pipe ramp and the simulated vehicle tracking the plan in open-loop mode. b) Motion captured test vehicle performing the same manoeuvre.

[1]N. Keivan, S. Lovegrove and G. Sibley are with the Department of Computer Science, George Washington University, Washington DC, `nima|slovegrove|gsibley@papercept.net`

Considering that the planning, MPC and model learning systems all utilise a model of the system, a unified system could be conceived to utilise the same model to perform all three tasks. The main contribution of this paper is such a system encompassing planning, control and online model learning using a unified, simulation-based model and operating in real time. We use a singular boundary value solver in all three cases in conjunction with cubic curvature polynomials for parameter reduction [7] . This allows accurate planning and control in full 3D environments and allows the learning of physical model parameters such as wheel radius, steering angle ratios and friction coefficients. Ground-truthed experimental evidence is also presented showcasing the results of the system planning over waypoints on undulating terrain and subsequently tracking the trajectory on a high speed nonholonomic robotic platform with on-line model learning.

II. METHODOLOGY

The different components of the planning and control system rely on a unified boundary value solver which produces a control law in order to navigate the robot between the start and goal 6DOF poses. For the purposes of this paper, we have implemented a parameter reduction and boundary value solver to plan for and control a nonholonomic remote control robot through high speed trajectories on undulating terrain. This formulation relies on a good initial guess for the steering and acceleration commands between two waypoints w_1, w_2 each parametrized as $[x, y, z, p, q, r, v]$ where v is the desired velocity with which the robot should reach the 6DOF coordinates of the waypoint. The optimization is facilitated with an initial guess utilising cubic curvatures for steering, and a linear velocity profile between waypoints.

A. Dynamics Model

The centrepiece of the system is the dynamics model. We use the Bullet Physics Engine [1] to simulate the dynamics of a vehicle with nonholonomic constraints on 3D terrain. A multithreaded framework allows the full use of modern multicore processes resulting in quick simulations for finite-difference based optimization. Traditionally, the state transfer function is defined as:

$$\check{x} = f(x, u, p)$$

Where x is the current state, u is the control input, and p defines the model parameters. In the case of the numerically integrated Bullet Physics model, the state transition is defined as:

$$x_{t+1} = F(x_t, u, p)$$

Where $F(x, u, p)$ encompasses the entirety of the dynamics of the vehicle and interaction with the terrain. In general terms, this function can be replaced with any simulation system resulting in an update in the state, given the control inputs, previous state and model parameters.

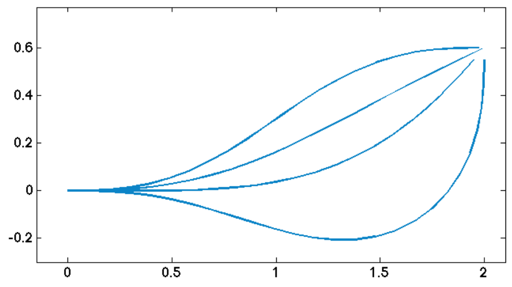

Fig. 2. Integration of cubic curvature polynomials in Cartesian space between $[0, 0, 0, 0]$ and $[2, 0.6, \pi, 0]$ with varying values of π

B. Parameter Reduction

The boundary value solver used relies on the reduction in control space dimensionality with the use of a control law. In the proposed system, we have employed cubic curvature polynomials [7] as a means to parameter reduction.The trajectory curvature is parametrized as a function of the travelled distance in the following form:

$$\kappa = a + bs + cs^2 + ds^3 \tag{1}$$

Where a is the starting curvature, b, c, and d are the cubic polynomial coefficients and s is the distance travelled along the trajectory. Individual polynomials are constrained using the endpoints coordinates $[x, y, \theta, \kappa]$. To obtain the cubic parameters necessary to reach the desired endpoint, a precomputed lookup table is employed followed by a Gauss-Newton optimization using the analytical Jacobian of the polynomial. Figure 2 shows an example of cubic curvature polynomials integrated in 2D Cartesian space. However since the planner operates in 3D space, we project the curvature polynomial onto a 2D plane, with a normal which is linearly interpolated between the normals of the two waypoints as shown in Figure 3. This allows the 2D curvature to better estimate the control law that will guide the vehicle and resolves singularities from waypoints perpendicular to the ground plane.

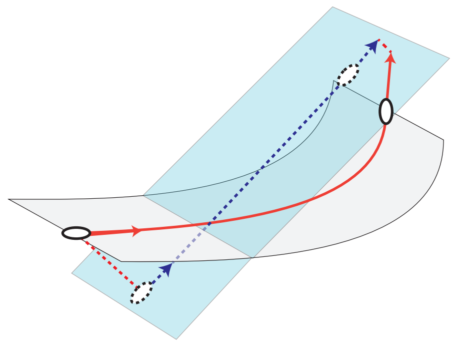

Fig. 3. The projected 2D trajectory (dotted blue) between two 3D waypoints on a curved manifold. This serves to better estimate the trajectory between the waypoints as well as eliminate singularities when projecting waypoints which are perpendicular to the ground plane.

C. Model Compensation

The linear velocity profile used between waypoints employs a constant acceleration model. However, due to the underlying physics-based vehicle model, this simple acceleration control law does not constitute a good initial guess. To improve upon this guess, several compensation factors are utilised to mitigate the extraneous influences introduced by the terrain and vehicle dynamics. Compensations are applied iteratively after each physics model update. This allows folding in detailed terrain information such as slope and also simulated vehicle parameters such as suspension force and extension. Furthermore, the underlying constant acceleration model remains valid once all other factors are compensated for.

1) Gravity Compensation: The constant acceleration model used between waypoints is by definition unable to account for terrain slope and undulation effects. We have implemented a simplified compensation model which accounts for the axial forces imparted by wheel interaction with inclined terrain (See Fig. 4a). The position of the wheels as well as the corresponding contact normal is obtained after each simulation step, and used to compensate the acceleration model for the following step.

2) Steering Compensation: Figure 4b shows the axial force component imparted as a result of the front wheel deflection during cornering. This force results in significant deceleration during tight turns and is compensated for in a similar fashion to gravity compensation at the end of each simulation step.

3) Friction Compensation: Friction compensation is undertaken iteratively similar to previously discussed factors. At each timestep, the friction forces on each wheel are calculated by the physics model. This is then used to offset the constant acceleration model accordingly. We have opted to use a simple friction model based on static/dynamic coefficients of friction, and the normal forces imparted on the springs. This information is readily available from the physics-model at each simulation step.

D. Boundary Value Solver

The boundary value optimization is performed by minimizing the trajectory cost C which we have defined as the 6 dimensional residual between the destination waypoint and the simulation endpoint. The optimization is performed by first solving a Gauss-Newton iteration with line search, and if the error norm is not reduced, a coordinate descent step is performed if possible. The Jacobian of the forward simulation is defined as:

$$J = \begin{pmatrix} \frac{\partial c_1}{\partial p_1} & \cdots & \frac{\partial c_1}{\partial p_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial c_n}{\partial p_1} & \cdots & \frac{\partial c_n}{\partial p_n} \end{pmatrix} \tag{2}$$

Where p_n is a control law parameter (such as a curvature polynomial coefficient) and c_n is a cost parameter. In the presented implementation, the cost is calculated as projected back onto the 2D plane of the cubic curvature polynomial (See Fig. 3) and is parametrized as $[x, y, \theta, v]$. Each column $\frac{\partial c_1}{\partial p_j} \cdots \frac{\partial c_n}{\partial p_j}$ of J is calculated by pushing forward the dynamics model using a set of control parameters p with perturbations $\pm\epsilon$ along dimension j. This computation is accelerated by the use of a multithreaded forward physics model, solving for all dimensions of the Jacobian simultaneously. The Gauss-Newton delta (δp) is then calculated by Cholesky factorization as follows:

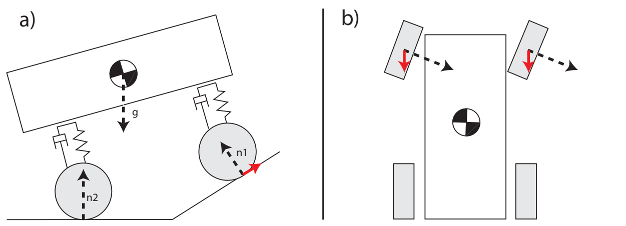

Fig. 4. a) Axial forces (in red) resulting from wheels on inclined terrain. b) axial forces (in red) resulting from front wheel steering deflection.

$$\begin{aligned} J^T J &\rightarrow R^T R \\ R^T y &= J^T b \\ R \delta p &= y \end{aligned}$$

Where b is the vector of residuals calculated by running the current parameters p and obtaining the endpoint error(s). The validity of the assumption of quadratic convergence made by this optimization is dependent on many factors including interactions with the terrain and the dynamics model. After obtaining the Gauss-Newton δp, we perform a multithreaded line-search step by pushing forward the physics model simultaneously with several scaled values of δp.

$$p_{n+1} = p_n + \lambda(\delta p)$$

Where $\lambda \leq 1$ the a scaling factor. If none of the scaled values of δp improve upon the error norm, we perform a coordinate descent if possible, by using the best norm obtained when calculating the Jacobian (Eq. 2) by finite-differences. The optimization ceases if the either the error norm is improved past a certain threshold, or if we are in a local minimum as indicated by the inability to perform a coordinate step to reduce the error norm.

E. Real-Time Control

In this section we present an MPC-like real-time control scheme based on fast replanning to account for inaccuracies and extraneous influences. Similar to MPC based control systems, our approach is formulated by constantly optimizing the trajectory ahead of the vehicle by solving new *control plans* which provide a viable control law from the vehicle's current position, to a point on the trajectory further ahead. As part of the holistic approach, we have used the same boundary value solver previously described to plan between waypoints, in creating the control plans. Due to the unified

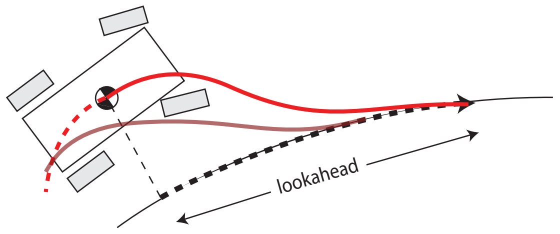

Fig. 5. Replanning-based control. The active control plan's (red) initial curvature matches that of the vehicle's current path curvature. Each control plan is optimized to plan to a point ahead on the trajectory based on a pre-define lookahead time.

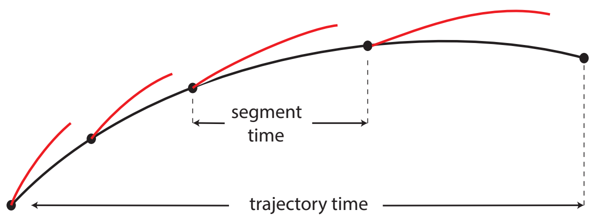

Fig. 6. The observed trajectory (black) shown with segments and their respective simulation results (red). The disparity between the simulation endpoint and the segment endpoint is the residual which is minimized in the optimization. If the residual is zero, the model perfectly matches the real vehicle's performance.

underlying steering control law, the control plans tend to converge back onto the original trajectory, thereby avoiding the pitfalls of follow-the-leader trajectory trackers which are prone to diverge if the target vehicle is set too far ahead, or oscillate if the target vehicle is too close. However, in order to achieve this behaviour, the starting curvature (denoted by the constant a in Eq. 1) of the control cubic curvatures has to match that of the vehicle's instantaneous path curvature as shown in Figure 5. This is also a requirement for smooth steering between control plans, as each will start with a curvature equivalent to that of the vehicle's current path. Consistent starting curvature is guaranteed by setting the constant a before the 2D optimization which solves Eq. 2. However, if the initial curvature required is too high, the 2D cubic required to solve the control plan might be infeasible. In these cases, the control system falls back to an initial curvature of zero to solve for a feasible control plan. Ideally, an alternative control law formulation would enable control plans that converged rapidly with any choice of initial curvature.

Furthermore, each control plan takes a small amount of time to optimize via the boundary value solver. During this time, the vehicle will be following the previous control plan. Our implementation includes a timestamp which is used to smoothly interpolate between plans as they become available. The control plans also constitute an any-time algorithm, as the optimization simply improves upon the quality of the initial guess with each iteration. The more time given to the algorithm, the higher the accuracy of the endpoint will be. It must be noted that each control plan is a valid set of controls that should ideally converge the vehicle back onto the trajectory in open-loop mode, given an accurate vehicle and terrain model. Therefore real-time control is established if the time taken to optimize tractable control plans is less than the lookahead time, allowing constant replanning without ever exceeding the bounds of a single control plan.

F. Online Model Learning

The quality of the control and planning provided by the aforementioned system relies significantly on the quality of the underlying model. Since a full physics-based model is used in the optimization, the number of parameters which could be adjusted rules out the possibility of manual tuning. We propose an optimization based learning system which is formulated almost identically to the control and planning systems, and which serves to tune select parameters in the model to match those of the real vehicle. The proposed methodology involves observing the vehicle and also the control commands given to it for a period of time, and optimizing the underlying physics model to replicate the observed behaviour, given the same control commands. We formulate the optimization by splitting the observed trajectory into segments as shown in Figure 6. Each segment is then simulated and a Jacobian formed as per Eq. 2 but where p_n is a model parameter which is changed by $\pm\epsilon$. Due to the nature of the optimization, we can obtain J^TJ and J^Tb for the trajectory directly as follows:

$$J^TJ = \sum_{i=0}^{n} J_i^T J_i$$

$$Jb = \sum_{i=0}^{n} J_i^T b_i$$

Where J_i is the Jacobian if the ith segment, n is the total number of segments, and b_i is the error vector of the ith segment which we have defined as the disparity between the observed trajectory and the final position of the simulated vehicle (See Fig. 6). We then apply the resulting δp to the model and repeat the process. As per the planning and control optimization formulations, the learning system implements a Gauss-Newton and line search stage which is followed by a coordinate descent stage if needed. The optimization ends when either a sufficiently small norm is obtained, or if a local minimum is detected. The learning system is implemented as an on-line algorithm allowing continued refinement of model parameters.

III. Results

The planner, control and learning systems were experimentally validated in a motion captured environment, using a terrain model which was 3D scanned using a Microsoft Kinect sensor combined with the motion capture system and a fusion algorithm. Due to the simulation-based model, any method of obtaining 3D terrain data could be used to simulate the dynamics. This includes real-time acquisition using cameras, laser scanners and/or dense tracking and mapping (DTAM) methods. In our experiments the waypoints were manual placed over the terrain in order to put the vehicle through desired manoeuvres including straight tracks, curves, steep inclines and jumps.

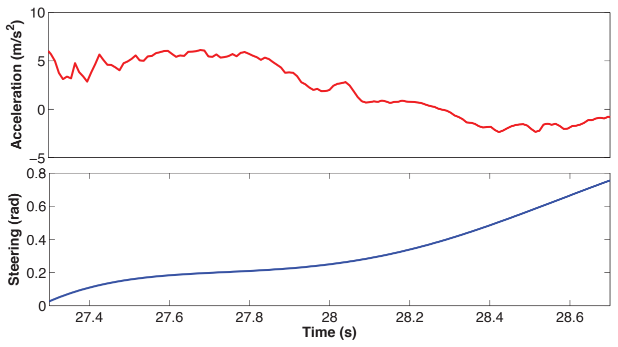

Fig. 7. Acceleration (red) and steering (blue) commands generated by the planner after optimization for the ramp manoeuvre depicted in Fig. 1. Note the gravity compensation during the uphill and downhill sections resulting in acceleration and braking forces.

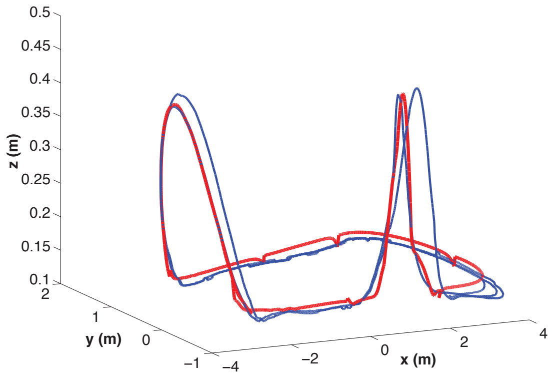

Fig. 8. Motion capture results (blue) of the control system running on a planned trajectory (red) with a small jump (center) and quarter-pipe ramp manoeuvre (left). The model was tuned using a combination of the learning system and manual adjustments.

A. Boundary Value Solver

Due to the quality of the initial guess, the boundary value solver successfully resolves feasible trajectories between waypoints. Furthermore the underlying model follows these trajectories precisely in open-loop control. However the choice of waypoints heavily influences the success or failure of the planner, as is expected. Figure 1 shows a plan generated between two waypoints on a scanned 3D model of a quarter-pipe ramp and Figure 7 shows the resulting acceleration and steering commands. Gravity compensation can be seen in Figure 7 and is vital to the feasibility of this plan as the vehicle needs to accelerate uphill and decelerate downhill in order to maintain velocity. The local planner can fail if the waypoints are poorly positioned or their velocities chosen improperly, for example if two waypoints are placed either side of a wall.

B. Real-Time Control

The real-time controller was tested on a trajectory including a small jump and sharp turn over a quarter-pipe ramp. Figure 8 shows the resulting vehicle path (blue) compared to the planned trajectory (red). It must be noted that the model used in this experiment was tuned using a combination of the learning system and manual adjustments. Manual adjustments were necessary as model-learning was not performed in high acceleration scenarios such as jumping, and small adjustments to the learned parameters was required to optimize the performance. The results obtained show that the vehicle is capable of accurately tracking the trajectory even in challenging manoeuvres, however as seen in Figure 8 , divergence can still be observed in the case of jumps where the steering is ineffective. This has the tendency to disrupt the real-time controller, as the boundary value Jacobian becomes invalid if changes in control input do not adequately perturb the endpoint of the control plan. This is the case when the vehicle is airborne.

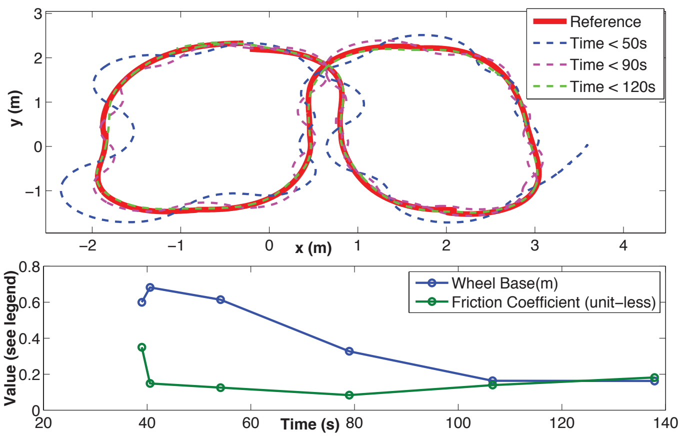

Fig. 9. Motion capture results of the control system (top graph) running on a planned figure-eight trajectory (red) while performing online model learning. The trajectories at various times show the improvement in tracking as the model parameters converge (bottom graph) to stable values.

C. Online Model Learning

The learning system was tested on a figure-eight trajectory over two model parameters: friction coefficient and wheel base. These model parameters, while not being all encompassing in their influence over the model have significant impact on the steering and acceleration response of the vehicle. They were chosen to test the learning system's response to deviations from the trajectory. While these parameters have underlying physical units, it is not expected that the values obtained from the model learning will reflect these values. This is due to the existence of unknown factors such as servo command coefficients, that will ultimately change the physical bases of the parameters. Nevertheless, the learned wheel-base parameter was observed to converge reliably to a value close to the actual vehicle wheelbase of $0.28m$. However, it is expected that learned values will bring the underlying model closer to the real-world vehicle, and therefore improve planning and control. Figure 9 shows the results of model-learning on a flat figure eight trajectory at different time intervals. It is evident that as the model parameters for wheel base and friction change, the adherence of the vehicle to the intended trajectory is significantly improved.

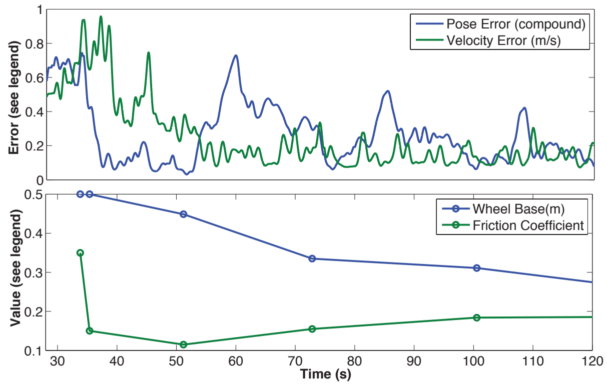

Fig. 10. Aggregated 6 DOF pose (blue) and velocity (green) error as a function of time performed on the figure-eight trajectory of Figure 9. The effect of the convergence of parameters can be seen on the repeated trajectory error pattern.

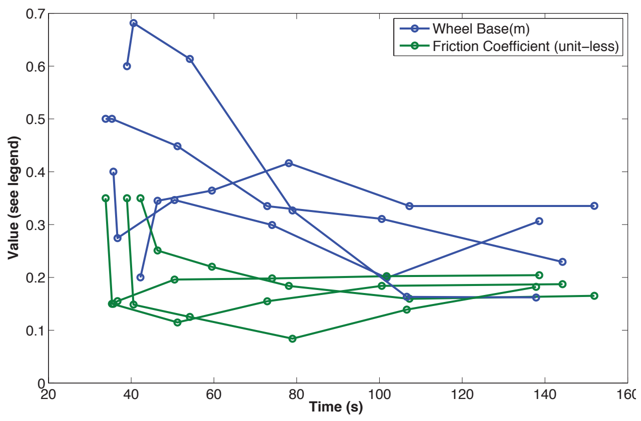

Fig. 11. Model parameter evolution during on-line learning for multiple experiments on the figure-eight trajectory of Figure 9.

Figure 10 shows the aggregated position and velocity error for a single run on the figure-eight trajectory, while performing model learning. It can be seen that the repeated error pattern around the trajectory monotonically decreases as the parameters converge. This also corresponds to a significant visible reduction in overshoot and improved tracking. Figure 11 shows the parameter convergence over multiple runs on the figure-eight trajectory and with different starting values for the parameters. It can be seen that the parameters, while not perfectly converging to the same point every time, tend to arrive at similar values.

IV. CONCLUSIONS

We have presented a holistic solution to local planning, real-time control and model learning which uses a unified simulation-based underlying physics model, folding in complex vehicle and terrain dynamics. The presented solution uses cubic curvature control laws to reduce the dimension of the control space while employing iterative compensation to deal with extraneous effects such as friction, terrain slope and steering deceleration. The solution was experimentally validated on a motion-captured vehicle and shown to execute manoeuvres on banked terrain and small jumps in real-time. However the real-time control system is still susceptible to disruption over jumps, and the model learning system has not been fully validated with a large number of parameters. Further experiments should also evaluate the ability to produce physically accurate model parameters. However, both systems have been shown to work well in practice and have produced good results in a challenging setting. Future work will be aimed towards testing the system on more challenging terrain, validating the model-learning with more parameters and implementing global planning solutions to place waypoints using the same holistic approach.

REFERENCES

[1] Bullet physics engine. `http://bulletphysics.org`. [Online; accessed July-2012].

[2] Anil Aswani, Patrick Bouffard, and Claire Tomlin. Extensions of learning-based model predictive control for real-time application to a quadrotor helicopter. In *ACC 2012, to appear*, Montreal, Canada, June 2012.

[3] Thomas Howard, Colin Green, and Alonzo Kelly. Receding horizon model-predictive control for mobile robot navigation of intricate paths. In *Proceedings of the 7th International Conferences on Field and Service Robotics*, July 2009.

[4] Thomas M. Howard and Alonzo Kelly. Optimal rough terrain trajectory generation for wheeled mobile robots. *Int. J. Rob. Res.*, 26(2):141–166, February 2007.

[5] Tor A. Johansen and N-Trondheim Norway. Computation of lyapunov functions for smooth nonlinear systems using convex optimization. *Automatica*, 36:1617–1626, 1999.

[6] S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. In *Robotics: Science and Systems (RSS)*, Zaragoza, Spain, June 2010.

[7] Alonzo Kelly. Trajectory generation for car-like robots using cubic curvature polynomials. Pittsburgh, PA, June 2001.

[8] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

[9] Mihail Pivtoraiko and Alonzo Kelly. Kinodynamic motion planning with state lattice motion primitives. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011.

[10] Neal Seegmiller, Forrest Rogers-Marcovitz, and Alonzo Kelly. Online calibration of vehicle powertrain and pose estimation parameters using integrated dynamics. In *IEEE International Conference on Robotics and Automation*, May 2012.

[11] R. Tedrake. LQR-trees: Feedback motion planning on sparse randomized trees. In *Proceedings of Robotics: Science and Systems*, Seattle, USA, June 2009.