2020 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
AAIR TECHNICAL SESSION
AUGUST 11-13, 2020 - NOVI, MICHIGAN

Automated Tuning and Calibration for Unmanned Ground Vehicles

Nate Bunderson, PhD[1], David Bevly, PhD[2], Austin Costley[1], William Bryan[2], Gregory Mifflin[2], Cristian Balas[3]

[1]Autonomous Solutions, Inc., Petersboro, UT
[2]Department of Mechanical Engineering, Auburn University, Auburn, AL
[3]CCDC GVSC, Warren, MI

ABSTRACT

A critical and time-consuming part of commissioning an unmanned ground vehicle (UGV) is tuning and calibrating the navigation and control systems. This involves selecting and modifying parameters for these systems to obtain a desired response. Tuning these parameters often requires experience or technical expertise that may not be readily available in a time of need. Even the simple task of measuring the mounting location of the sensors introduce opportunities for user error. In addition, the tuning parameters for these systems may change significantly between UGVs. These challenges motivate the need for automated tuning and calibration algorithms to set parameters without the interaction from a user. This work presents automated tuning and calibration approaches for UGVs.

Citation: N. Bunderson, D. Bevly, A. Costley, W. Bryan, G. Mifflin, C. Balas "Automated Tuning and Calibration for Unmanned Ground Vehicles", In *Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium* (GVSETS), NDIA, Novi, MI, Aug. 11-13, 2020.

1. INTRODUCTION

Preparing an unmanned ground vehicle (UGV) for operation typically requires calibration and tuning. These processes include setting parameters that represent physical characteristics (i.e. mass, wheelbase, sensor mounting locations) of the vehicle (calibration), and parameters that determine the behavior of the localization and control algorithms (tuning). Establishing a base-line calibration and tuning parameter set often requires specialized knowledge and the manual modification of parameters to achieve a desired performance. This process scales poorly and can be time-consuming for fleets of autonomous vehicles.

This work presents a method for auto-calibrating sensor mounting locations with respect to a known point on the vehicle. This is accomplished by leveraging a machine learning technique that uses a dynamic vehicle model and a known series of inputs to emulate the IMU signals at the known point.

Additionally, two methods for autotuning the controller parameters to improve system performance are presented. The first method is an algorithm that searches a defined parameter space to improve the model of the system behavior. A model predictive control strategy is employed so the improved model will directly improve the controller performance. The second method is an

adaptive control law that mimics the behavior of a human driver that will adjust to uncertainty in the system model.

This paper presents the auto-calibration method in Section 2. Then the model predictive path controller is presented in Section 3. Section 4 presents the autotuning algorithm and provides results for estimating parameters for the steering model and dynamics model of an automated ground vehicle. Section 5 then provides the formulation and results from the controller based on human behavior. Lastly, Section 6 gives an overview of the conclusions drawn from this research.

2. SENSOR MOUNTING ESTIMATION AND CALIBRATION

The goal of the sensor calibration module is to locate and orient the sensors on an autonomous vehicle relative to a known control point. This will allow for more accurate sensor integration. In addition, it will enable flexibility in sensor configuration. Once the calibration routine is performed, the sensors on the vehicle may be positioned at the discretion of the team operating the autonomous vehicle.

A calibration routine can be run to locate a sensor relative to another sensor. However, assuming full modularity of the sensor configuration, no sensor will be placed at the known control point. To locate a sensor relative to the control point, the output of a sensor fixed at that location will need to be emulated.

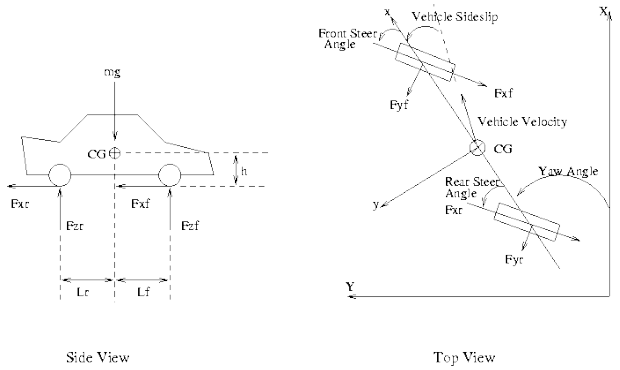

Figure 1 Vehicle Model Diagram for derivation of the Dynamic Bicycle Model

2.1. Solution Approach

To calibrate the sensors relative to a fixed control point, an inertial measurement unit will be emulated for that point. To approximate the output of the inertial measurement unit, a vehicle model will be developed for the control point. Without knowledge of an exact description of the translational and rotational accelerations, the model will be constructed using experimental data.

A machine learning approach was taken for this task. A recurrent Gaussian kernel network [1] was built to perform the IMU emulation. Neurons are added sequentially to form a single hidden layer of Gaussian kernels. In each training cycle, the algorithm adds a neuron at the location of maximum residual error. The network continues to train until it is stopped by a cross-validation algorithm. The cross-validation algorithm prevents overfitting of the training dataset.

When operated in open-loop mode, the network will predict the target acceleration values, a_x, a_y, and a_z, based on the vehicle's current position in the state-space of past input values.

A rigid-body transformation is then used to estimate the lever arm from the control point to the IMU physically attached to the vehicle. A standard least squares algorithm can then be applied to extract the true lever arm parameters from the noisy lever arm predictions.

$$\hat{a}_{sensor} = \boldsymbol{R}\,(\,\hat{a}_{cp} + \alpha \times r + \,\omega \times \omega \times r) \qquad (1)$$

Equation (1) defines the rigid-body transformation equation used to estimate the lever-arm, r, between the control point and an IMU located on the vehicle.

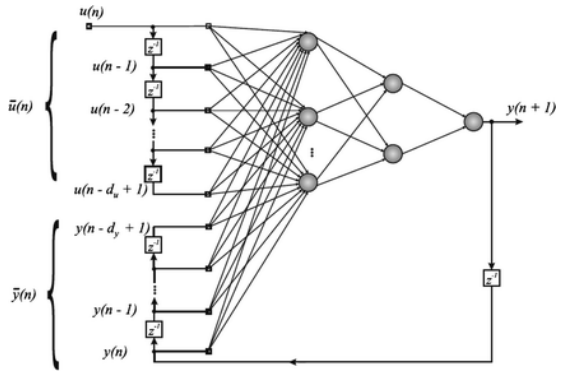

Figure 2 Recurrent Neural Network Architecture of the Same Type as the calibration algorithm

2.2. Sensor Calibration Results

The calibration algorithm was trained and tested on simulated data obtained from the Carsim 9 engine. The simulated dataset was produced based on a set of sine sweep steer angle inputs. Velocity in the inertial frame was kept constant, while the other inputs to the neural network could vary freely in the simulation.

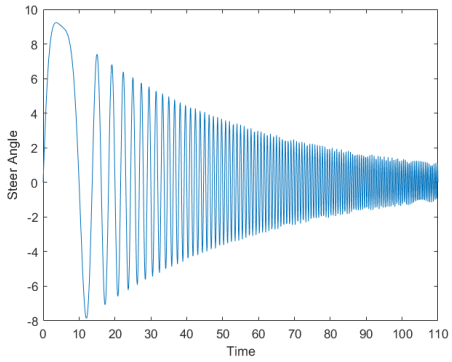

Figure 3 Sine sweep steer angle input, used to generate simulated training data

Seven measured quantities were used as inputs to the neural network model: steer angle (δ), longitudinal velocity (v_x), lateral velocity (v_y), vertical velocity (v_z), pitch rate (ω_θ), roll rate (ω_φ), and yaw rate (ω_ψ). All of these values can be reliably measured with common sensors and are not dependent on the location of the sensor on the vehicle.

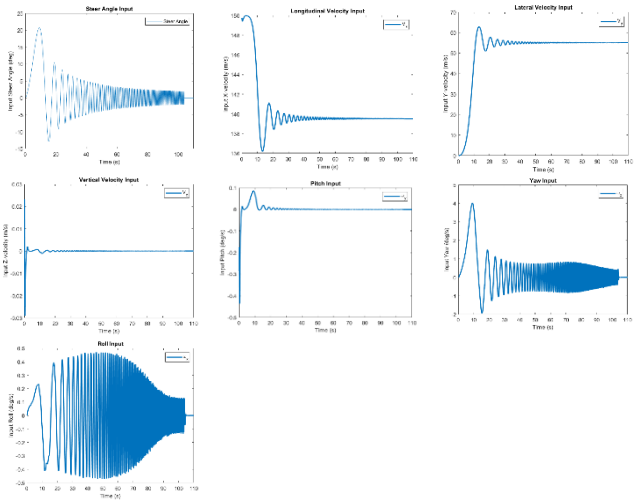

Figure 4 The seven input time series, taken from a single training run

From these input time series, the neural network model generated a prediction of each of the three translational accelerations, a_x, a_y, and a_z. Given the rigid-body assumption, the angular velocities should remain the same anywhere on the vehicle. Hence, the complete IMU model is given by these three predicted values.

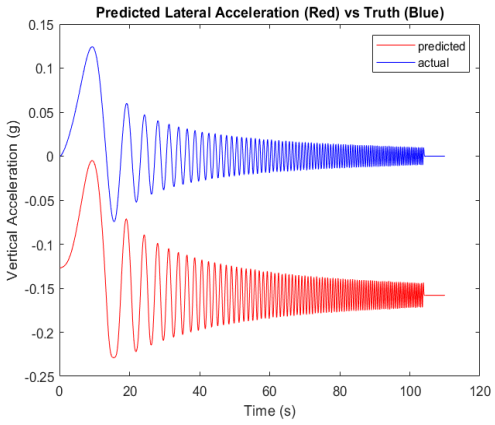

Figure 5 Lateral Acceleration predicted by the neural network model differs from truth by a constant value

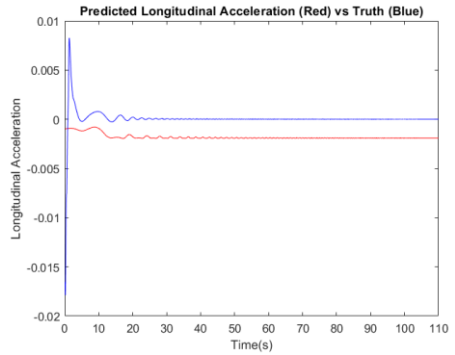

Figure 6 Longitudinal Acceleration also differs from truth by a small constant. The initial spike is not detected

The lever arm between the control point and another simulated IMU is then estimated. A small constant bias can be noted in both plots. The origin of this bias is unclear, but an investigation of the neural network reveals that the issue lies with the way that the mean value of the time series is approximated. Despite the errors in the model predictions, the lever arm estimates are accurate so long as the transformation matrix remained well-conditioned. The noise in the predictions appears to be a result of issues with the transformation matrix. However, a simple least-squares algorithm will filter it out to a large extent.

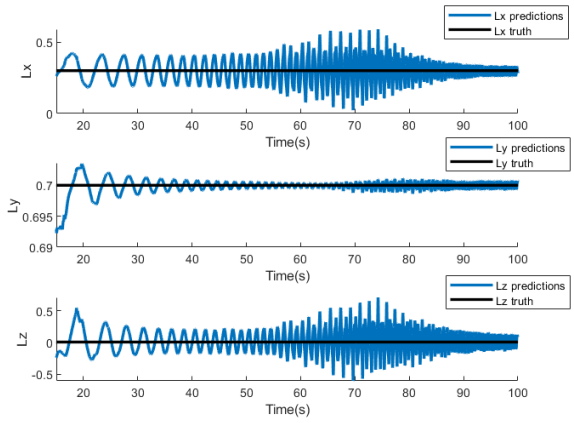

Figure 7 Estimated Lever arm time series in blue, and true lever arms in black

However, if the transformation matrix is ill-conditioned, whether because some variables are not observable, or because the angular accelerations derived from the measured angular velocities are inaccurate, then the lever arm estimates are degraded. L_x, the estimate in the longitudinal direction, is strongly affected. Deviation from truth of nearly a meter has been observed. The other two parameter estimates are less strongly affected.

The simulated results indicate that, given enough training data, and a well-conditioned transformation matrix, the algorithm can effectively estimate the lever arm between an IMU and a fixed control point on an autonomous vehicle. The other sensors can then be calibrated relative to the IMU using known techniques.

3. GRAFTERPLUS – MODEL PREDICTIVE PATH CONTROLLER

To showcase the autotuning algorithm presented in Section 4, a model predictive path controller (MPPC) called GrafterPlus was developed. A model predictive controller (MPC) uses a model of the system to be controlled to predict the behavior given a series of inputs. An optimization problem is formulated to minimize a cost function to determine an optimal series of inputs to track a given command signal [2].

In the context of path control, a dynamic or kinematic model of a ground vehicle is used to predict the system behavior and provide a trajectory for the vehicle to follow. A cost function penalizing cross-track and heading errors is formulated and the error is computed along a trajectory. The optimization routine determines a series of steer angle inputs that minimize the error along the trajectory. The first input from the optimal series of inputs is used as the next command to the vehicle. This process is repeated at every control interval.

MPC techniques have been used for decades [3]. Due to the high computational cost of this method, they have historically been restricted to systems with very slow dynamics. However, the increase in processor speed in recent years has made this method of control more attractive for a wider variety of problems.

The GrafterPlus controller has three stages, namely, propagate, plan, and predict. These stages are described as follows:

Propagate: Propagate the current vehicle state forward to obtain a vehicle state estimate after a given delay.

Plan: Create graft path to connect vehicle state from Stage 1 to point along desired path. The connection point is determined by the look-ahead parameter, d_l.

Predict: Use vehicle model to calculate the curvature command that will result in an average steering velocity (over a specified time horizon) required by the graft path in Stage 2.

This process is shown graphically in Figure 8, where Stage 1 is represented by the red dashed line, and Stage 2 is represented by the blue dashed line. The look ahead parameter, d_l, is shown below the desired path. Stage 3 is not explicitly depicted but it includes computing a curvature command due to the planned path in Stage 2.

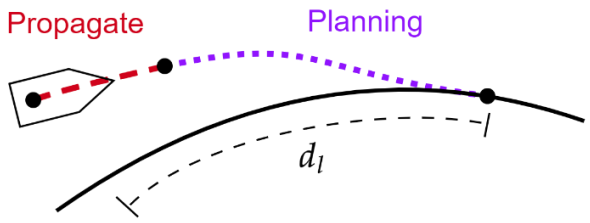

Figure 8 Graphical representation of GrafterPlus algorithm. The first and second stages are shown in red and blue, respectively. The third stage is to compute the curvature command associated with the graft path from Stage 2.

The vehicle model used in the Predict stage of GrafterPlus is the dynamic bicycle model described in Section 4.2. This model includes cornering stiffness parameters that strongly influence the lateral dynamics. Selecting this parameter to reflect the true cornering stiffness of the vehicle greatly influences the accuracy of the path controller. Figure 9 shows the results from two simulations of Grafter Plus with different cornering stiffness parameters. The simulated vehicle had a cornering stiffness of 250 N. The figure shows that when the parameter was correctly set in Grafter Plus the off-path error was significantly reduced. This motivates the need to autotune such parameters to accurately model the system behavior and improve the performance of the path controller.

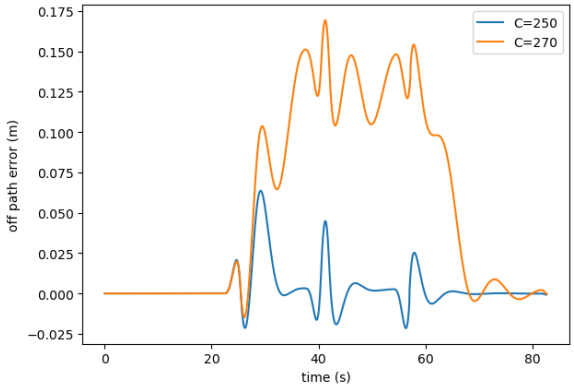

Figure 9 Off path error results for simulations of Grafter Plus with different corning stiffness parameters. The cornering stiffness of the simulated vehicle was 250 N. The plot shows that correctly setting this parameter results in improved performance.

4. AUTOTUNING PATH CONTROL

An autotuning algorithm was developed for an MPPC that uses a model of the system dynamics to simulate the step response of the system. A series of discretized (step) inputs is provided to both the model and the vehicle. The response of the model and vehicle are compared, model parameters are adjusted, and the process is repeated for each combination in the parameter space. The parameters associated with the smallest modeling error are then used in the controller.

The path controller presented in Section 3 is a good candidate for autotuning as it has very few parameters and the parameters generally relate to physical quantities of the ground vehicle. This section presents the autotuning approach for two models used in the Grafter Plus path controller. First, the steering model of the system is modeled as a second order system where the damping ratio and natural frequency are estimated. Second, a dynamic bicycle model is used to model the lateral

dynamics of the vehicle and the cornering stiffness parameter in the model is estimated.

Figure 10 shows an example of two system step responses. The blue curve represents the true response of the system and the black curve represents the response of the modeled system. The error is calculated as the area between the curves. The objective of the autotuning algorithm is to modify parameters of the modeled system to minimize this area.

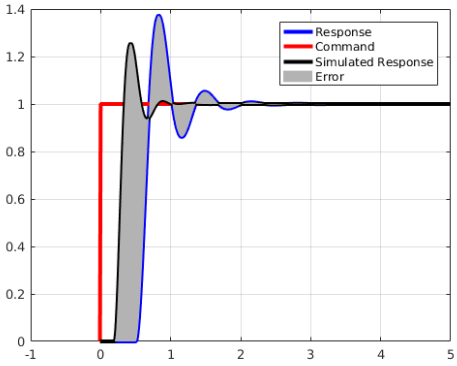

Figure 10 Example system response where the error is the area between the actual system response and the simulated (or estimated) system response.

The general autotuning algorithm used for the steering model and bicycle model autotuning is given in Algorithm 1. This algorithm was found to have the best results if the input signal were discretized to provide a series of step responses that could be compared with the step response of the model given a specific parameter set. With the input signal discretized into a series of steps, the true response of the system is stored in a "bin" for each command. The algorithm then loops through the combinations of parameters defined in the parameter space and simulates the step response of the model for each binned input. The error is computed as the accumulated difference between the system response and the modeled response. The parameter set associated with the smallest error is determined to be the best parameter set for modeling the system.

Algorithm 1 Autotuning algorithm used for both steer model, and bicycle model parameter autotuning.

Autotuning Algorithm		
1:	*P* = { parameter space}	*(set parameter space)*
2:	min_error, p_opt	*(initialize variables)*
3:	**for** p : *P*	
4:	**for** b : bins	*(loop through bins)*
5:	m = SimModel(p, b.cmd)	*(simulate model)*
6:	t = GetTrueOutput(b)	
7:	error += ComputeError(t, m)	*(compute error from truth)*
8:	**if** error < min_error	
9:	min_error = error	
10:	p_opt = p	*(store best parameter set)*
11:	**end if**	
12:	**end for**	
13:	**end for**	

It remains to define the models used in the autotuning algorithm. The second order steering model is defined in Section 4.1 which relates a steering angle command to vehicle curvature. Then the dynamic bicycle model is defined in Section 4.2.

4.1. Steering Model

The steering model is assumed to be a second order system parameterized by a damping ratio, ζ, and natural frequency, ω_n, given by

$$P(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n + \omega_n^2} \qquad (2)$$

where the input and output of the system are the setpoint and feedback of the vehicle curvature, respectively. The natural frequency and damping ratio determine the response of the system. Figure 11 shows the second order system response to a variety of damping ratio and natural frequency values.

Second order systems are commonly used to model higher-order systems. This is an attractive approach for the autotuning algorithm because by changing a couple of parameters a large variety of system responses can be achieved.

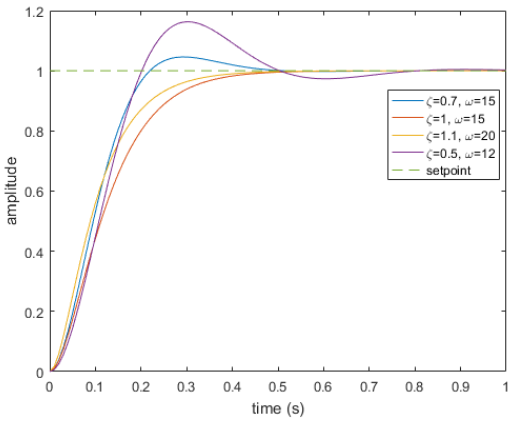

Figure 11 Second order system example for varying values of damping ratio and natural frequency

The geometry of the bicycle model of a ground vehicle shows that the curvature of the vehicle is approximately proportional to the front wheel angle. To account for this approximation and non-linearities in the steering system, the relationship between steering wheel angle and curvature is modeled as a 3rd order polynomial. Figure 12 shows an example of sampled curvature measurements in response to steer angle commands. The blue circles represent the curvature measurements and the red line is a best-fit polynomial used to represent the mapping between steering wheel angle and curvature.

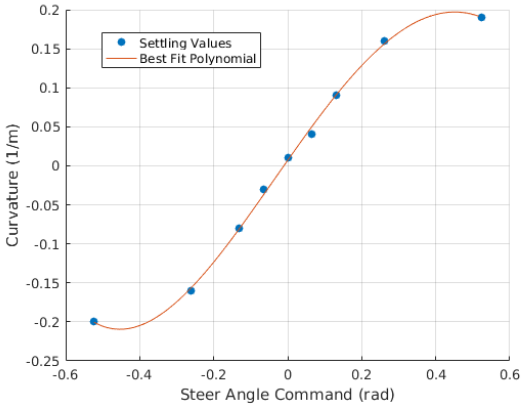

Figure 12 Steer polynomial example that relates steer angle to curvature.

In addition to the parameters in a second order system, the autotuning algorithm is also able to estimate a pure delay, *d*, from command to steering response. Thus, the three parameters estimated by the autotuning algorithm are ζ, ω_n, and *d*. The full steering model is shown in Figure 13 which illustrates the signal path of steer angle command to modeled curvature.

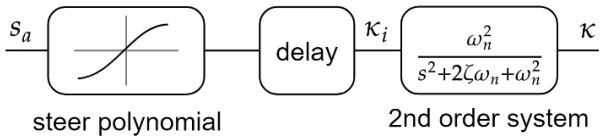

Figure 13 Steer model that takes steering wheel angle (hand wheel angle) and converts to curvature

The first step of the autotuning algorithm is to determine a parameter space. Table 1 provides an example parameter space for the steering model of an automated ground vehicle. This parameter space is very large and should account for a variety of ground vehicle types, however, it is left to the designer to determine appropriate ranges for the specific system of interest.

Table 1 Parameter space for steer model autotuning.

	Min	Max	Δ	Unit
d	0.05	1	0.05	s
ω_n	2	20	1	rad/s
ζ	0.1	2	0.1	-

Figure 14 shows the results of the autotuning algorithm for the steer model parameters. The blue line is the system response to a series of step inputs (the command values can be inferred by the shape of the response curves). The red curve is associated with a poor estimate of the system response, whereas the green curve (which tracks the system response very well) is the response of the model using the parameters determined by the autotuning algorithm. These results show that the autotuning algorithm can accurately estimate the steer model parameters.

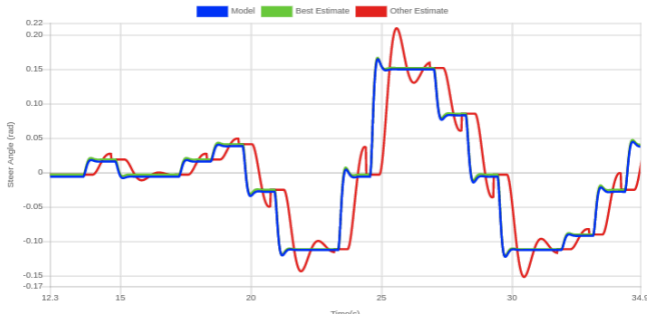

Figure 14 Steer model autotuning results.

4.2. Dynamic Lateral Bicycle Model

Inherent to a model predictive control strategy is a model of the system dynamics. This model is used to predict the behavior of the system given a series of inputs. An optimization routine is used to minimize the error between the commanded signal (path) and the predicted system behavior by modifying the series of inputs. The first input of the optimal solution is provided to the vehicle and the rest are discarded. The process is repeated at every control interval. This control strategy has been shown to be effective as a path controller for autonomous ground vehicles [4].

Typically, there are discrepancies between the system and the model used in MPC. An autotuning algorithm can be used to further refine the model and improve the accuracy controller.

In Grafter Plus, a linearized bicycle model [5] [6] has been chosen to model the lateral dynamics of a ground vehicle. The dynamics of this model are given by

$$\begin{bmatrix} \dot{r} \\ \dot{V}_y \end{bmatrix} = \begin{bmatrix} -\frac{a^2 C_{\alpha f} + b^2 C_{\alpha r}}{I_z V_x} & -\frac{aC_{\alpha f} - bC_{\alpha r}}{I_z V_x} \\ -\frac{aC_{\alpha f} - bC_{\alpha r}}{mV_x} - V_x & -\frac{C_{\alpha f} + C_{\alpha r}}{mV_x} \end{bmatrix} \begin{bmatrix} r \\ V_y \end{bmatrix} + \begin{bmatrix} \frac{aC_{\alpha f}}{I_z} \\ \frac{C_{\alpha f}}{m} \end{bmatrix} \delta \quad (3)$$

where r is the yaw rate, V_y is the lateral velocity, a and b are the distances between the CG to the front and rear axles, $C_{\alpha f}$ and $C_{\alpha r}$ represent the cornering stiffness of the front and rear tires, m is the mass of the vehicle, and I_z is the moment of inertia about the positive z-axis. Most of these parameters are well known or easy to measure. An exception to this is the cornering stiffness parameters which are difficult to estimate and often have a high degree of uncertainty. Thus, the cornering stiffness parameters are an ideal candidate for autotuning.

The linearized bicycle model was implemented in the Grafter Plus path controller and the autotuning algorithm was used to tune the cornering stiffness parameters of the vehicle using Algorithm 1. The results of a simulated test is shown in Figure 15, where the curvature of the actual model is represented by the green line. The red line is the curvature of the model with the autotuned cornering stiffness (18 N). The dark blue and light blue lines are the curvature of two other models from the parameter space provided to the autotuning algorithm.

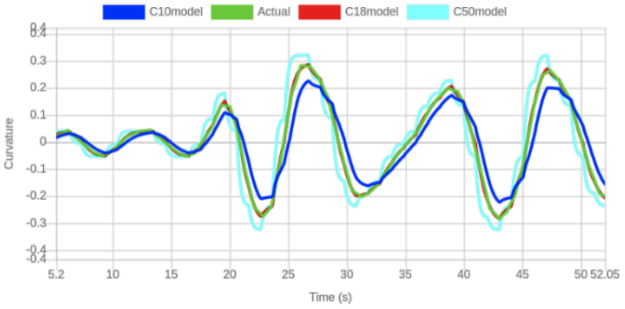

Figure 15 Vehicle curvature during model parameter autotuning tests in simulation. The green line represents the curvature of the actual system. The red line represents a model with the autotuned cornering stiffness (15). The blue lines are models using other parameters checked by the autotuning algorithm.

5. MANUAL DRIVING ADAPTIVE CONTROL (MDAC)

The motivation behind Manual Driving Adaptive Control (MDAC) is to design a controller that behaves similarly to a human. This involves two main components. First, MDAC should follow the same driving characteristics of a typical human driver, such as similar path following dynamics and smoothness. Second, it should be able to adapt to a wide variety of vehicles and vehicle types, just as a human can drive anything from a large truck to a

small sports car with the same basic knowledge. In this work, the lateral aspect of the MDAC is investigated.

To achieve these desired characteristics, various controller architectures were explored and tested. Dynamic model-based controllers can perform well on a specific platform if tuned well [7], but they require very accurate vehicle models and do not do well when applied to a different platform. Robust controllers can guarantee stability margins for a wider range of systems; however, they cannot handle extremely large levels of uncertainty or guarantee the desired performance [8]. Direct adaptive control, such as model reference adaptive control, was therefore a logical next step due to abilities to handle large amounts of uncertainty while guaranteeing stability and performance [9]. Nevertheless, when implemented in high fidelity simulation and on a real vehicle, it was found to perform much worse than expected. This was due poor estimates of certain vehicle states needed for the adaptive controller and higher order dynamics not captured in the reference model.

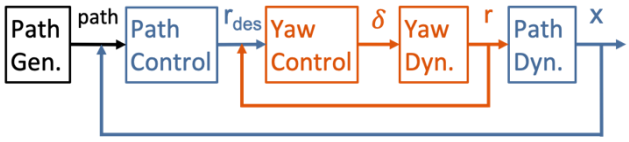

Figure 16 MDAC Architecture

Figure 16 shows the architecture that was chosen for the MDAC. It employs a cascaded approach with the path dynamics in the outer loop (blue) and the yaw dynamics in the inner loop (orange). These path dynamics can be made to match the path dynamics of typical human driving behavior, which is mostly independent of vehicle type. Then the yaw dynamics, which are often unknown, are dealt with separately. This system is implemented using Robotic Operating System (ROS) in C++ and Python.

Generating the desired reference path to follow is the first step. In this work, two methods were used for this: vision-based lane detection and GPS waypoint following. The vision-based system uses a camera to detect lane markings and would be used in a lane keeping system. This was implemented in simulation in Gazebo and on the vehicle real-time. The GPS waypoint path generation uses a GPS receiver to record the desired reference path. This would be used on or off road to follow a predetermined route or to follow another connected vehicle, such as in vehicle platooning [4]. This reference path is rotated and translated into the frame of the controlled vehicle and sent to the path following node.

The path following node then calculates the desired yaw rate as

$$r_{des} = r_{FFW} + r_{FB} \tag{4}$$

to follow the reference path. This is based on a feedforward term (to follow the curvature of the path) that is calculated by multiplying the longitudinal velocity of the vehicle by the curvature of the path. Thus, the feedforward term is given by

$$r_{FFW} = \kappa(s) * V_x,\ \kappa(s) = \frac{d\theta}{ds} \tag{5}$$

and a feedback term (to correct deviations from the path) as

$$r_{FB} = -K_p * e_{LA}. \tag{6}$$

The error in the feedback equation is the lookahead error. To mimic human driving behaviors, the lookahead distance is adapted based on vehicle speed and path profile. The lookahead distance acts like damping in the system and is increased proportional to longitudinal speed. This replicates how a driver on the interstate looks further ahead than they do when driving slow through a parking lot. Additionally, the lookahead distance is increased as the curvature of the path decreases. This is similar to how human drivers should look further ahead on a straight road than a tight twisty one.

The desired yaw rate from the path following node is sent to the yaw controller. While various

controller types were tested for this piece, the best results were from inverting a simple kinematic model. This model only relies on knowing a wheelbase length and the current vehicle speed, which makes it easily adapted to different platforms. Figure 17 shows how well the yaw controller is able to track the commanded yaw rates during a simulation run around a test track in Gazebo.

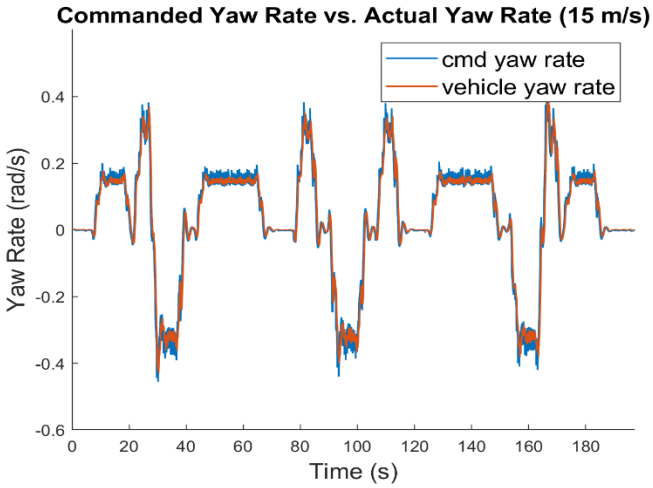

Figure 17 Yaw Response using Kinematic Steering Model

The steer angle calculated by the yaw controller is sent to the test vehicle through the Controller Area Network (CAN) bus, utilizing the original equipment manufacturer steering motor.

5.1. Results

The Manual Driving Adaptive Control was first tested in simulation using a test track Gazebo environment and the Dataspeed MKZ Gazebo model. The simulation uses the vision-based lane detection for the reference path generation. Figure 18 shows the results for a full lap of the Gazebo test track. The plot displays a higher lateral error than the true path error because it was calculated using the lateral lookahead error from the camera at the closest path point in the camera's field of view. The distance to the closest path point is approximately 8 meters. Although this is not the truth value, it shows the way the vehicle is generally behaving relative to the path and is able to be used for comparison purposes. Qualitatively, the controller kept the vehicle within the lane bounds at all times during the run.

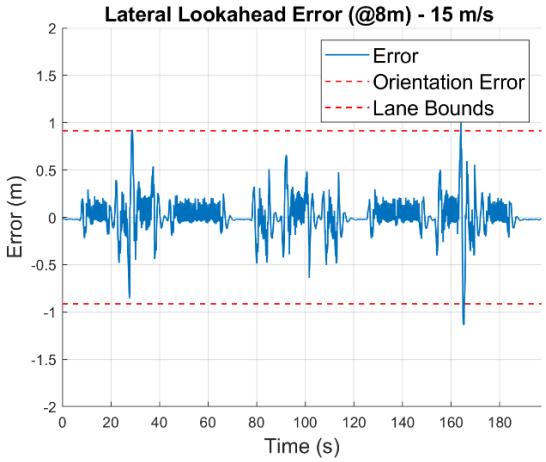

Figure 18 Simulation Lookahead Error Results

Real world testing was done using a double lane change path of GPS waypoints. The double lane change was chosen because it is a high dynamic maneuver and can be easily compared to a human driver. This test was run at various speeds and the results can be seen in Figure 19 and Figure 20. The tests were done at various speeds, but 5 m/s and 10 m/s are shown here. Each plot shows three consecutive runs.

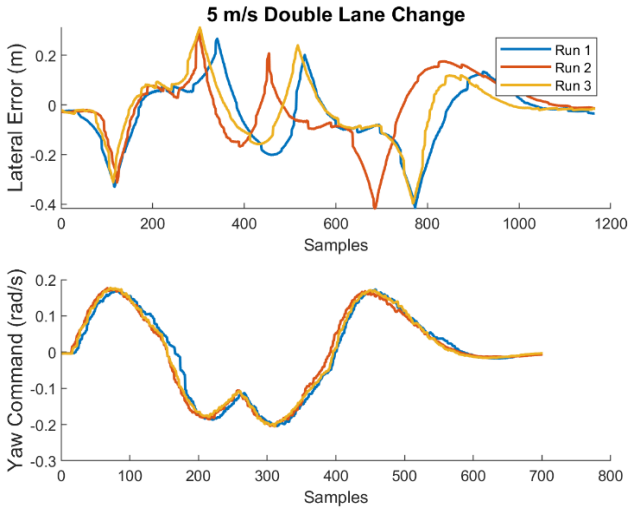

Figure 19 5 m/s Double Lane Change

Figure 19 shows that for 5 m/s, the lateral error is less than half a meter throughout the maneuvers. Quantitatively, the steering was extremely smooth and precise.

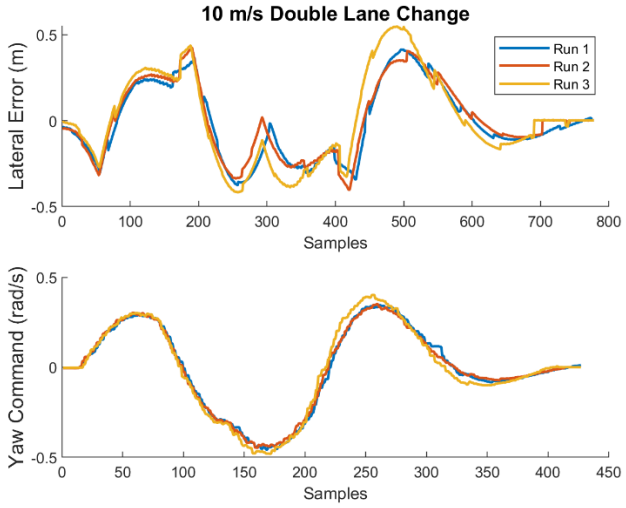

Figure 20 10 m/s Double Lane Change

At 10 m/s the max lateral error remained around half a meter for all three runs as shown in Figure 20. Again, the steering was exceptionally smooth.

Overall, Manual Driving Adaptive Control (MDAC) was successful at maintaining low lateral errors at a variety of speeds, while also mimicking the path following behavior of a human driver. Combined with the smoothness of steering actuation, MDAC provides another possible approach to lateral vehicle control, without having to do extensive manual tuning.

6. CONCLUSION

This work presents methods for automating the calibration and tuning of an autonomous vehicle. The calibration method leveraged a recurrent Gaussian kernel network to estimate the sensor mounting location of an IMU. Prior to operation the neural network would need to be trained on data from the vehicle of interest. Once trained, this process could be used to estimate the mounting location of the IMU anywhere on the rigid body.

Two methods for autotuning controller gains were presented. The first method involves a brute force parameter search approach attempting to minimize differences between the responses of a reference model and the vehicle. Results were provided indicating accurate model estimation, which was in turn used in an MPPC to improve path control. The second method was an adaptive controller called MDAC, which attempts to mimic the behavior of a human driver adapting to a new vehicle. MDAC was able to follow a reference path with about 0.5 m of maximum lateral error. MDAC provides another approach of controlling an uncertain UGV without relying on a dynamic model.

7. REFERENCES

[1] N. Benoudjit, C. Archambeau, A. Lendasse, J. Lee and M. Verleysen, "Width optimization of Gaussian kernels in Radial Basis Function Networks," in *European Symposium on Artificial Neural Networks*, Belgium, 2002.

[2] J. B. Rawlings and D. Q. Mayne, Model predictive control: theory, computation, and design, vol. II, Madison, WI: Nob Hill Publishing, 2017.

[3] D. Q. Mayne, J. B. Rawlings, C. V. Rao and P. O. M. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789-814, 2000.

[4] F. Borrelli, P. Falcone and T. Keviczky, "MPC-based approach to active steering for autonomous vehicle systems," *International Journal of Vehicle Automation Systems*, vol. 3, no. 2, pp. 265-291, 2005.

[5] P. Polack, F. Altche, B. d'Andrea-Novel and A. d. La Fortelle, "The Kinematic Bicycle Model: a Consistent Model for Planning Feasible Trajectories for Autonomous Vehicles?," in *IEE Intelligent Vehicles Symposium*, 2017, 2017.

[6] J. Kong, M. Pfeiffer, G. Schildbach and F. Borrelli, "Kinematic and Dynamic Vehicle Models for Autonomous Driving Control Design," in *IEEE Intelligent Vehicles Symposium*, Seoul, 2015.

[7] N. R. Kapania and J. C. Gerdes, "An Autonomous Lanekeeping System for Vehicle Path Tracking and Stability at the Limits of Handling," in *International Symposium on Advanced Vehicle Control*, Tokyo, 2014.

[8] D. Gu, P. Petkov and M. Konstantinov, Robust Control Design with MATLAB, Glasgow: Springer, 2005.

[9] T. Yucelen, Model Reference Adaptive Control, New York City: Wiley, 2019.

[10] J. Ward, P. Smith, D. Pierce, D. Bevly, P. Richardson, S. Lakshmanan, A. Argyris, B. Smyth, C. Adam and S. Helm, "Cooperative Adaptive Cruse Control (CACC) in Controlled and Real-World Environments: Testing and Results," in *Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium*, Novi, 2019.

[11] K. Åström and T. Hägglund, PID controllers: theory design and tuning, NY: Instrument Society of America, 1995.

[12] A. O'Dwyer, Handbook of PI and PID controller tuning rules, Singapore: World Scientific Publishing, 2009.