Obstacle Avoidance using Model Predictive Control: An Implementation and Validation Study Using Scaled Vehicles

Author, co-author (Do NOT enter this information. It will be pulled from participant tab in MyTechZone)

Affiliation (Do NOT enter this information. It will be pulled from participant tab in MyTechZone)

Abstract

Over the last decade, tremendous amount of research and progress has been made towards developing smart technologies for autonomous vehicles such as adaptive cruise control, lane keeping assist, lane following algorithms, and decision-making algorithms. One of the fundamental objectives for the development of such technologies is to enable autonomous vehicles with the capability to avoid obstacles and maintain safety. Automobiles are real-world dynamical systems – possessing inertia, operating at varying speeds, with finite accelerations/decelerations during operations. Deployment of autonomy in vehicles increases in complexity multi-fold especially when high DOF vehicle models need to be considered for robust control. Model Predictive Control (MPC) is a powerful tool that is used extensively to control the behavior of complex, dynamic systems. As a model-based approach, the fidelity of the model and selection of model-parameters plays a role in ultimate performance. Hardware-in-the-loop testing of such algorithms can often prove to be complex in its design as well as in its implementation. Therefore, in this paper, we explore a less-used deployment toolchain that combines the power of ROS (Robot Operating System) in intra-robot communication with motors and sensors with the rich library of controller models in Simulink Real-Time. In particular we explore this rapid-control-prototyping in real-time to deploy Model Predictive Control for Obstacle Avoidance on a ROS-based scaled-vehicle. We found that this framework is user-friendly and contains great potential for educational and research-bed deployments – with a short development and deployment time that can fit neatly in one semester.

Introduction

Contemporary automotive obstacle-avoidance and motion-planning research builds upon the decades-old research in wheeled-mobile-robot (WMR) systems[1]. Some of the earliest (and simplest) WMR algorithms feature some variation of bug algorithms [2, 3], or potential-field techniques to map the shortest possible path to the goal while avoiding the obstacle [4, 5]. Strategies that are more complex involve a layered approach, such as splitting the task between low-level and high-level planners[6, 7]. A local (or lower level) planner may be involved with more real time control of steering and velocity making minute decisions, trajectory corrections or collision avoidance based on short-range sensing data, whereas a global planner would be in-charge of the big picture. A global planner would be the path-planning algorithm that would provide the heading and direction of the overall goal position or a reference trajectory.

Many of the early methods focused computational efficiency (especially for real-time deployments) primarily for wheeled robots operating at slower speeds. Hence motion-planning approaches focused on model-free approaches or employed simplified kinematic models (e.g. bicycle models) However, the transition from wheeled-mobile-robots deployments (lower-speed) to real-world automotive deployments now requires consideration of the vehicle dynamics, due to the inertias, range of operational speeds and finite acceleration/deceleration capabilities. Fortuitously, modeling, analysis and control of real-world automobile systems has a rich history that can be brought to bear [1, 8].

Model Predictive Control (MPC) methods are a well-established class of control techniques developed for optimally controlling multivariable systems with constraints on plant and actuators (largely in the chemical-process industry). However, computational-complexity had restricted deployments to longer-time scale dynamical systems (such as chemical-processes) with centralized computational infrastructure. However, the need for improved dynamic-control performance of automotive-systems coupled with an improved understanding of benefits of MPC algorithms and increased mobile computational power have led to rapid growth of deployments in the automotive-applications [9] (from engine-control to vehicle-handling). In recent times, MPC techniques have been extended to empower dynamic-model-based motion-planning [10] – beginning with a nominal global reference path (or reference trajectory) derived from a global planner to provide a control action based on a prediction of the future state of the modelled vehicle. Borrelli et al [11] first proposed an MPC-based approach (NLMPC) for active steering on a reference trajectory for a double lane change maneuver by assuming that the global (reference) trajectory was pre-determined by an obstacle avoidance planner.. Lee et al [12] performed a comparative analysis on various techniques in simulation and then validated the efficacy of model predictive controllers as compared to PID and LQR based methods. Hrovat et al [9] performed a survey on the state-of-the-art techniques that use some or the other form of model based method in the automotive industry where they are used to develop active safety systems. All the work mentioned above concludes that there is a need to develop the testing infrastructure necessary to validate related work for the following reasons:

- Model based methods are growing in complexity.
- The controllers must be tuned to most of the situations that are likely to be encountered on the road.

- Increasing power of computational tools and improvements to workflows for designing and testing the algorithms is relevant to the future of the work mentioned above.

For the specific case of MPC deployments in small-scale vehicles, Thilén [13] explores the implementation of the ROFMPC on the ROS, which exports a MATLAB based MPC code as a node on the vehicle platform. Verschueren et al [14] implemented a non-linear MPC using the Acado toolkit on an RC car. While the viability and benefits of MPC-based motion-planning methods have been established, the increased algorithmic complexity and the need to gain experience with real-world real-time hardware-in-the-loop deployments remains a challenge (especially in an educational setting). The rich library of open-source control algorithms and short-deployment times of the ROS-Simulink Real Time framework make it an ideal candidate for such deployments moving forward.

Our research and educational program has been exploring autonomy performance testing with scaled Autonomous Remote Control cars (based on the F1tenth.org parts-list) [15]. In our previous deployments [16], student teams were successful in navigating around a closed racecourse at speeds of 10-15 miles per hour, using Simultaneous Localization and Mapping (SLAM) for situational awareness and basic collision-avoidance (stopping when obstacles are detected).

In the current manuscript, we discuss our further efforts for integration of a model-based rapid control prototyping pipeline (real-time MPC active steering controller for a double lane-change maneuver for obstacle avoidance) using these scaled autonomous "F1/10" RC cars. To this end, we leverage the tools provided by MATLAB, in particular, the Model Predictive Control Toolbox and the Robotic Systems Toolbox to provide a framework for rapid-prototyping of controllers in a simulation setting. Coupled with the Real-Time-Workshop toolbox, this enables hardware-in-the-loop testing and finally production-level development and real-time deployment to the target hardware that can support verification-and-validation efforts. The critical point of this work would not be the actual algorithm or controller design, but the tools used to combine multiple resources into a single toolchain that would enable the initial design and tuning of a model based controller in a visual friendly environment such as MATLAB-Simulink and quick deployment of the method onto ROS based hardware for further hardware in the loop testing and validation of the selected model based control technique on a scaled vehicle, all within an educational setting.

This paper is organized as follows. The next section outlines the development of the mathematical vehicle model and the controller strategy. The following sections talk about the practical realization, integration of different hardware and software layers, the test scenario development and constraints for the same and finally the physical test bed and our results.

Model Predictive Control Development

Model predictive control looks at the current state of the vehicle (steering angle and velocity) and estimates the future states of the vehicle. Based on this prediction, at reach time step, the performance index or the cost function minimizes the error between the future state and desired state (obtained from the reference trajectory generated from the global planner at time *t*) within the bounds of some operating constraints. This progressive horizon or desired states are the "look-ahead" for prediction of system states on which the control signals are optimally calculated for the succeeding time step. The constraints are based on the mathematical representation of physical operating bounds for the hardware system in use. These will be the maximum rate of change for steering as well as throttle (soft constraints) and the maximum and minimum values for the same (hard constraints).

Vehicle Model

The double-lane change MPC controller employed in this paper was inspired from the example model for obstacle avoidance provided by Mathworks [17]. It employs a simple kinematic bicycle model [18] which was modified to RC car dimensions (to enable subsequent testing on our scaled F1/10 testbed described in the next section). Figure 1 describes the model based on the following assumptions:

1) The vehicle travels at low speeds and longitudinal velocity is constant
2) No wheel slip, no tire forces
3) Steering only at the front wheels
4) The model lumps the left and right wheels into a single wheel on the axis along the center of gravity.

The equations of motion for the simple kinematic bicycle model are given as,

$$\dot{x} = cos(\theta)v$$

$$\dot{y} = sin(\theta)v$$

$$\dot{\theta} = (tan(\theta)/C_l)v$$

$$\dot{v} = 0.5T$$

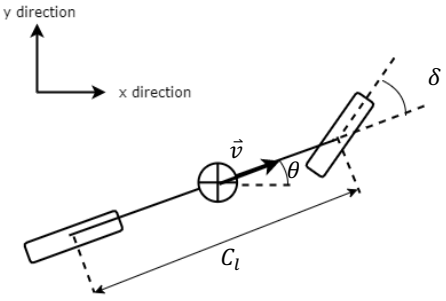


Figure 1. Simple kinematic bicycle model without tire forces.

Where x, y are the position coordinates of the center of mass of the car in the global frame, θ is the heading direction, v is the vehicle velocity and C_l is the length of the car. The state of the system is represented by $X = [x, y, \theta, v]^T$ and the control input, u, is the steering angle, δ. Since these are continuous and non-linear in θ, it is inferred that the cost function will require a non-linear MPC. To avoid this, the system was linearized about an operating point (at time t_0) when the vehicle is at the origin, with zero heading and constant velocity.

$$\dot{x} = -vsin(\theta_0).\theta + cos(\theta_0).v$$

$$\dot{y} = vcos(\theta_0).\theta + sin(\theta_0).v$$

$$\dot{\theta} = (tan(\delta)/C_l).v + (v(tan(\delta)^2 + 1)/C_l).\ \delta$$

$$\dot{v} = 0.5.T$$

Once the initial step is executed, the plant model is updated and linearized about the next operating point at time which are the states at its previous time-step. This requires a full state feedback to the plant model at each time-step and hence all states were kept observable. A zero order hold method discretizes the linear-continuous equations.

There are also physical constraints that need to be specified for the vehicle that are characterized as follows:

	Physical parameter	*Constraint type*
Steering angle	Positive lock	Hard constraint
	Negative lock	Hard constraint
Rate of change of steering angle	Positive rate	Soft constraint
	Negative rate	Soft constraint

Table 1: Constraint properties

The soft constraints allow for overshoot in both directions in case the look-ahead is too low and the desired trajectory is too aggressive. The hard constraints are provided to the MPC controller to represent the maximum steering angle bounds in both directions (full lock left and right) such that the optimization problem would always give a viable input command to the vehicle.

Controller Strategy

The MPC controller was built from the Model Predictive Controller Toolbox in Simulink. Again, our focus in this manuscript is to transform this development of model-based MPC controller into a deployable real-world platform that can support verification and validation. Hence, we present critical aspects of the development of the test setup (for completeness) but refer the reader to [16] for further information.

The Adaptive MPC (A-MPC) was found to have significant advantages over a linear MPC. It was more suited to handle the changing system dynamics due to the assumption of using throttle as a control input and velocity as a state. This is due to the sequential linearization that occurs in the plant model at every time-step. Implementation of A-MPC has implications with increased computation time [19], and thus the fixed time step chosen would need to be greater than the per-iteration computation time. Thus, the time-step for simulation was fixed at 0.02 s. A non-linear MPC might provide better performance but at the cost of higher computation times.

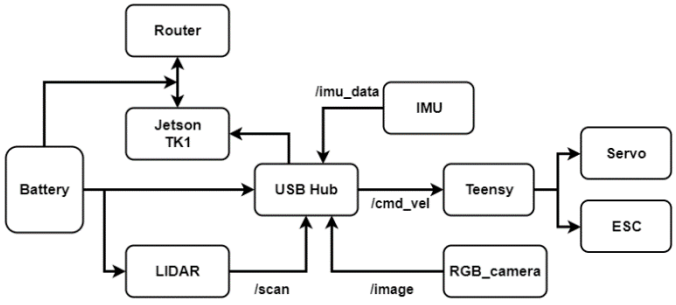
The manipulated variables are throttle (T) and steering angle (δ), whereas observed variables are x & y position of the vehicle, heading (θ) and vehicle absolute velocity (v). The observed variables have a certain weight attached to them as a priority for the controller to track. These individual weights alone do not make any physical sense, but the ratio of the weights is used as a control knob to tune the controller in simulation, and later in real-world HIL testing. The cost function weighs the x, y position and velocity tracking higher, while giving a zero weight to the heading of the vehicle allows its value to float based on changing control inputs for the steering angle (δ).

$$J = \sum_{t=1}^{h} W_x||x_t - x_r||^2 + W_y||y_t - y_r||^2 + W_v||v_t - v_r||^2 + W_\theta||\theta_t - \theta_r||^2$$

The reference trajectory is provided from the initial position of the vehicle right up to the goal point, which is in front of the obstacle within its lane. Within these reference waypoints, the controller only looks at the closest h points as parameterized as the prediction horizon, at each time-step. The controller then estimates the position of the vehicle for these future points and uses it to predict reference-tracking error for the future states. The controller was found to work best with a prediction horizon of 90 steps without compromising much in terms of computation cost. The control horizon was parameterized to 5-10% of the prediction horizon and was set to 5 steps, implying that the control inputs are optimized for only five steps.

SCALED "F1/10" Autonomous Remote-Control Car

The F1/10th platform created by O'Kelly et al [20] from University of Pennsylvania serves as our primary experimental platform. The main chassis of the vehicle is a TRAXXAS Ford Fiesta ST scaled 1/10th model remote-control car equipped with two motors. Digital wheel velocity command-inputs are provided to a stock Electronic Speed Control (ESC) which provides the requisite current to power a high RPM DC motor driving the rear-wheels through a differential. The second motor is a servo motor at the front for steering.



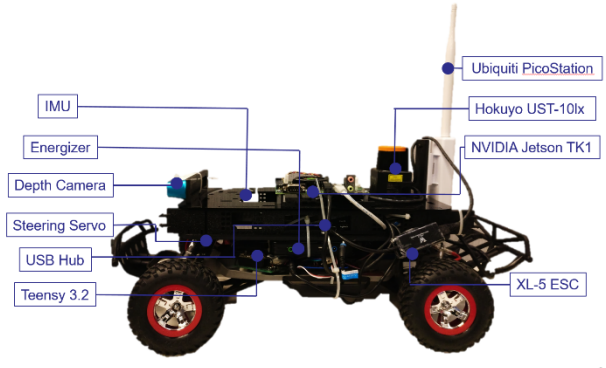


Figure 1: The systems and sensors of the test vehicle

The main computational board is an Nvidia TK1 running Ubuntu 14.0 installed with ROS Indigo. All on-board sensors are interfaced through ROS. The figure shown below depicts the connections between the onboard sensors and the computational platform. In this paper, the LIDAR is the only sensor employed from the available sensor stack.

Lateral velocity of the vehicle is for a given PWM signal to the ESC is non-linearly dependent on the battery level. The signal was calibrated every 10-15 runs to ensure constant velocity for each run of the controller.

The subscriber within the onboard processor has a buffer which stores the control commands and stacks them up until the previous PWM signals are sent out. This would cause a delay between the control commands sent by Simulink and the actual execution of the same on the vehicle. To avoid these problems, a last in first out strategy was implemented which reads the latest data in the buffer. This is implemented with a queue size of 1 in the subscriber, hence clearing the buffer at every step.

Software Interface

In this paper, we demonstrate a unique method of performing soft-real-time testing of a well-defined methodology for obstacle avoidance, i.e. Adaptive MPC. Usually, the obstacle avoidance logic is written in a generic programming language such as python or C++ and deployed directly as a ROS node on the vehicle. This approach requires a lot of hardware time, right from the start of tuning the controller and up to the final deployment of the method.

The technique that is demonstrated in this work allows the use of tuning tools within the simulation framework i.e. Simulink, such that the major part of the controller can be designed and tuned to the scaled vehicle specification without the presence of hardware. Later, this can be directly deployed to the ROS framework on the scaled vehicle, while allowing us to leverage the versatility and simplicity of toolboxes within MATLAB-Simulink to better visualize and understand the behavior of the robot, while also speeding up the process of directly comparing it with simulation results for the same scenario. The real-time portion that is implemented is a relatively new feature of the MATLAB-Simulink ecosystem that they call simulation pacing. This technique is used to synchronize the processes running in ROS (lower level control) with the processes running in Simulink (issuing control commands for vehicle behavior). This would have previously been difficult without the use of external mode compilation of the controller model in Simulink, or without the use of purpose built real-time processing hardware.

Implementing a controller on a hardware platform requires complex signal conversions, scaling, and the hardware capacity to handle the computational load on the mobile CPU. With the ROS+Simulink toolchain, the bulk of this computational load is shifted to a remote PC which is capable to simultaneously interacting with multiple ROS enabled vehicles at a time.

The input and output signals from the MPC need to undergo three levels of conversions and mapping before they can be read by the ROS enabled vehicle. The first is the scaling factor multiplied to the control input (steering angle) to scale it within the MPC object, such that a smooth control input can be provided to the vehicle. These steering angle values are converted from radians to degrees before sending it through the publisher. The final level of conversion happens within the Teensy board where the command input is converted to a PWM signal and is finally sent to the ESC and servo to drive the vehicle.

The MPC toolbox in Simulink [17] provided the basis for the implemented MPC design. The Robotic System Toolbox (RST) provided a bridge between the ROS enabled ego vehicle and Simulink to allow for transmission of control signals from Simulink and receiving sensor data from ROS.

The ROS publisher in Simulink is first populated with a blank message while only changing the part that is useful, such as steering. The message that the ROS system subscribes to is a topic named /cmd_vel which has linear and angular values in 3 dimensions. The steering output from the controller is filled into the blank message at the correct position using a bus assignment and published to ROS at every time-step.

The on-board system running ROS will then subscribe to this message and send a signal to a Teensy board which has been programmed to accept inputs from a serial port. Once this communication is done, the final implementation of the control signal is performed by the Teensy which will convert the value it receives into a PWM signal that finally drives the servo and DC motors through an electronic speed control module. The car moves forward in the next time-step and the sensor takes a measurement of position and orientation of the car. This signal is sent through the onboard processor and published to the ROS network using a VRPN streaming service. This service publishes a pose stamped ROS message that can be picked up by any system on the ROS network. In our case, we created a subscriber in the Simulink model that listens for these messages. Further, we break down the measurements into the x and y position and use it to form the updated states of the vehicle. The orientation data received from the ROS message is in quaternions and is converted using standard Simulink conversion blocks within the robotic system toolbox library, then added as the third state variable. The fourth state of the car is its absolute velocity which is kept constant for the first part of the hardware testing, hence eliminating the need for providing throttle as a control input to the vehicle.

This process is repeated at the rate of 50 Hz due to the fixed time-step specified in the model configuration pane within Simulink.

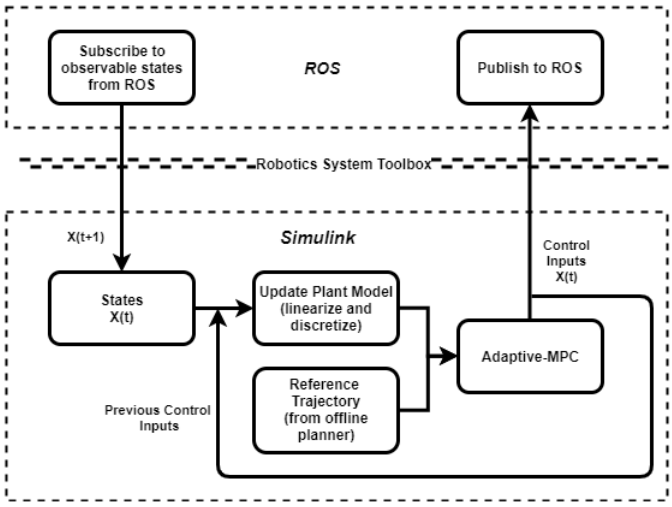


Figure 2: Complete system architecture

Test Scenario Development

In this paper, we assume a stalled lead vehicle which necessitates a double lane change maneuver for the follower (ego) vehicle to avoid

this obstacle. Our development of a suitable obstacle avoidance pipeline features global (off-board) and local (on-board) motion-planning components. An OptiTrack Motion-Capture (MoCap) system, meant to replicate a GPS in the indoor setting, provides position/velocity information of the ego vehicle. The perception framework (on the ego-follower vehicle) detects the obstacle and provides obstacle (lead vehicle) state information to the global planner within the ROS ecosystem. The global-planner then creates an obstacle-avoidance trajectory which serves as the reference trajectory input to our MPC algorithm. However, we note that was due to the limitations of the experimental testbed (as we will discuss later). The proposed framework can easily be extended to the case where the ego-vehicle is moving at a lower-speed in the current lane – since only the relative position/velocity of the ego-vehicle are necessary. In such cases, we can replace a global planner to generate the reference values and leverage the ability of MPC to handle constraints explicitly [17].

The complete system architecture (Figure 1) is composed of three parts – the perception framework for obstacle tracking, the global planner, the ROS and Simulink framework for communication of state information and the MPC model in Simulink Real-Time. The lowest level of control – converting output pose state from the MPC to PWM signals readable by the ESC and reading pose data (from the MoCap and IMU) also occurs over the ROS framework. The Robotics Systems Toolbox provides seamless integration of ROS with the MPC model in Simulink.

Test Bed

The controller was implemented on an autonomous scaled vehicle in an indoor test environment the Clemson (ICAR) campus in Greenville, SC. The A-MPC requires state information of the vehicle – position, velocity and heading – information that in the real world would be provided by a GPS. The test setup at CUICAR employed a ground-truth motion capture system OptiTrack to provide "GPS" like data in an indoor setting. The system (shown in Figure 4) comprises of 12 infrared cameras surrounding the 4.85 by 3.5 meter test space that track reflective markers fixed to the ego vehicle. The system has an accuracy of a few millimeters, which is significant when dealing in such small scales. The position & orientation information from the MoCap setup is supplied through a ROS-node to the global planner on the ego vehicle, which supplies trajectory information to the Simulink model. Velocity is assumed a constant.

Obstacle Avoidance

Perception

The ego-vehicle is equipped with a 2D Lidar (Hokuyo URG-04LX-UG01) for sensing obstacles. The obstacles are determined as a cluster of points based on distance from one another. This cluster is further processed to extract a cost-map of the surroundings. This is passed to an offline planner that generates the double lane change maneuver. In a traditional sense of obstacle avoidance, the vehicle's trajectory is conditioned on whether the control system is enabled and if it is in a certain range of an obstacle or not. This distance that is used to determine an imminent collision will change drastically in the real-world scenario, where we would need to consider the relative deceleration of the lead vehicle with respect to the ego car. In our case due to space and safe speed conditions for indoor experimentation, we assume a static obstacle and a detection range of 2 meters.

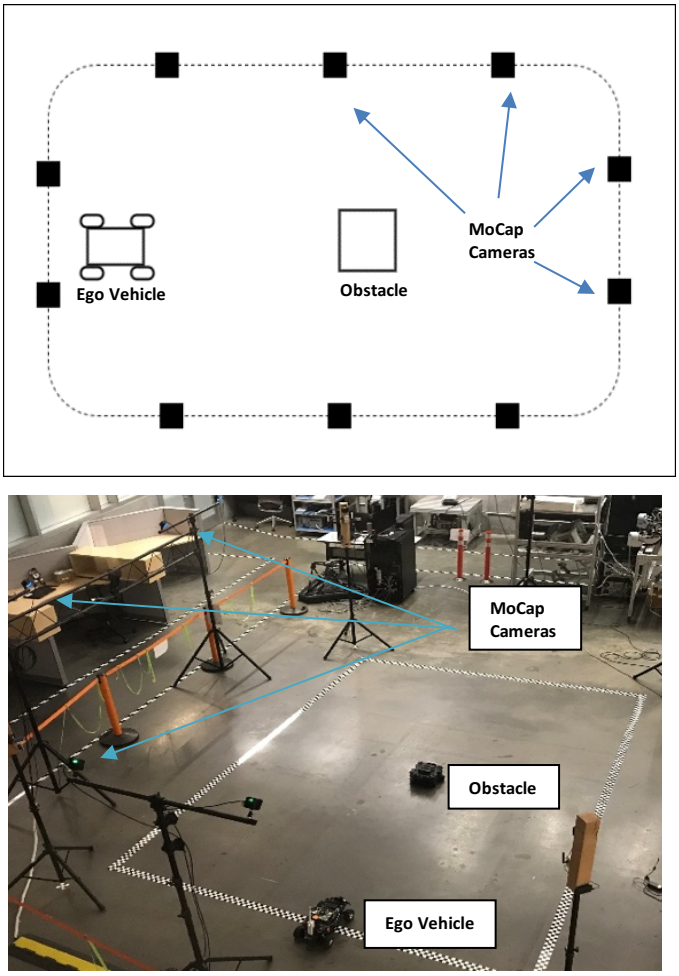


Figure 3: Test setup with OptiTrack

Global planner

The global planner is a rapidly - exploring random tree (RRT) algorithm that generates two random trees expanding from the start and the goal point to find a viable path for the vehicle to follow [21]. The two trees move towards one another using a simple greedy heuristic [22].

The next step is to generate a path from the ego vehicle to the point just outside the safe zone created around the obstacle. This is commonly referred to as configuration space creation in most path planning solutions where one must find the points within the space that is acceptable for the vehicle to traverse along. Using the extracted cost-map, the planner makes decisions based on certain factors such as inflation over boundaries, the dimensions of the robot etc. These are tuned to perform in a scaled vehicle setting and an indoor environment.

ROS + Simulink Communication Framework

The position and heading of the vehicle is measured using the OptiTrack system as mentioned earlier and streamed into the ROS network using a local VRPN server. This system records and streams the data in the format of a geometry_msg/PoseStamped ROS message. To stream this data into Simulink, we leverage the Robotic System Toolbox's features such as a block to subscribe and publish messages

to ROS from within Simulink. The messages that are published out of Simulink and into ROS is the control command that goes directly to the hardware. This command controls the steering of the vehicle and is populated by the control signals that the A-MPC controller sends as an output at each time-step. This ensures that the vehicle is receiving its control command directly from the A-MPC block.

Results and discussion

Initially the system was modelled and simulated in Simulink to verify the effectiveness of the controller. A simple sinusoid trajectory was provided as a reference trajectory to gauge the system response. Initially this was done in simulation where the time period was mapped to the length of the track and the amplitude was set to 1m. The weights assigned to the controller ensure that the MPC only prioritizes following the trajectory (the *x* and *y* position of the vehicle).

In figure 3, we see the difference between a mathematical model in simulation and the implementation of the same controller on physical hardware. We notice that the model in simulation follows the trajectory more smoothly than the HIL test. This difference can be visualized in figure 4 as the tracking error in simulation is far less as compared to the error in hardware testing.

These errors can be attributed to the simple kinematic bicycle model that is used as the plant. The assumptions imposed upon the system due to the simplistic nature of the model are that we do not know the tire forces, nor the slip angles of the vehicle. We also ignore the effects of roll, pitch and yaw on the physical system.

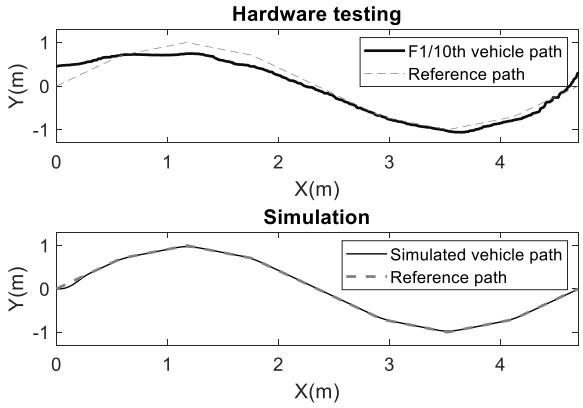


Figure 4: Trajectory comparison between simulation and hardware

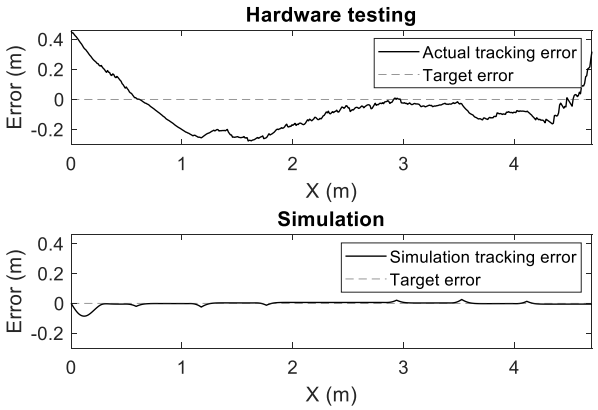


Figure 5: Error comparison between simulation and hardware

Summary

In this paper, we go through a few reasons why one cannot perform full scale autonomous testing while developing algorithms, nor can pure simulation be the right method of tuning said programs. We explore an alternative that lies between these two extremes (testing on scaled vehicles) that gives a lot more experience than pure simulation while maintaining a safe and repeatable environment for an academic setting. We also go through the pipeline for generating a vehicle trajectory for an advanced maneuver such as obstacle avoidance, and how that path can be executed by a lower level controller, using a model-based approach.

Developing the model in simulation was the first task that we undertook and was necessary to perform for initial results over which we could benchmark its performance. The link between MATLAB-Simulink (for rapid development solutions) and ROS (widely used distributed-computing platform) is one of the major interfaces that was required to successfully deploy the controller to hardware in an efficient manner. Using RST to perform this inter-framework communication, we went on to the next step of the project which was execution on the actual test vehicle. These results show us that even-though the model works well in simulation, there is a lot of scope for improvement when deployed on hardware.

The next steps would be to use the MATLAB code-generation capabilities to generate production level code to be deployed and run directly on the real-time target machine. Along with running the code purely for obstacle avoidance, this could be integrated with other controllers that rely on other sensor data (like a camera) to detect and perform other tasks (such as lane keeping) or a RADAR to perform ACC.

References

1. R. Rajamani, Vehicle dynamics and control: Springer Science & Business Media, 2011.
2. I. Kamon, E. Rivlin and E. Rimon, "A new range-sensor based globally convergent navigation algorithm for mobile robots," in Proceedings of IEEE International Conference on Robotics and Automation, pp. 429-435, 1996.
3. V. J. Lumelsky and T. Skewis, "Incorporating range sensing in the robot navigation function," IEEE Transactions on Systems, Man, and Cybernetics, vol. 20, no. 5 pp. 1058-1069, 1990.
4. J. R. Andrews and N. Hogan, "Impedance control as a framework for implementing obstacle avoidance in a manipulator," M. I. T., Dept. of Mechanical Engineering, 1983.
5. Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," in Proceedings. 1991 IEEE International Conference on Robotics and Automation, pp. 1398-1404, 1991.
6. J. F. Canny and M. C. Lin, "An opportunistic global path planner," Algorithmica, vol. 10, no. 2-4 pp. 102-120, 1993.
7. J.-P. Laumond, M. Taïx and P. Jacobs, "A motion planner for car-like robots based on a mixed global/local approach," in EEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications, pp. 765-773, 1990.
8. T. D. Gillespie, Fundamentals of vehicle dynamics vol. 400: Society of automotive engineers Warrendale, PA, 1992.

9. D. Hrovat, S. Di Cairano, H. E. Tseng and I. V. Kolmanovsky, "The development of model predictive control in automotive industry: A survey," in 2012 IEEE International Conference on Control Applications, pp. 295-302, 2012.
10. C. E. Garcia, D. M. Prett and M. Morari, "Model predictive control: theory and practice—a survey," Automatica, vol. 25, no. 3 pp. 335-348, 1989.
11. F. Borrelli, P. Falcone, T. Keviczky, J. Asgari, et al., "MPC-based approach to active steering for autonomous vehicle systems," International Journal of Vehicle Autonomous Systems, vol. 3, no. 2 pp. 265-291, 2005.
12. J. Lee and H.-J. Chang, "Analysis of explicit model predictive control for path-following control," PloS one, vol. 13, no. 3 p. e0194110, 2018.
13. E. Thilén, "Robust Model Predictive Control for Autonomous Driving," 2017,
14. R. Verschueren, M. Zanon, R. Quirynen and M. Diehl, "Time-optimal race car driving using an online exact hessian based nonlinear MPC algorithm," in 2016 European Control Conference (ECC), pp. 141-147, 2016.
15. University_of_Pennsylvania. F1tenth Build. URL: http://f1tenth.org/build.html.
16. A. T. Raman, V. N. Krovi and M. J. Schmid, "Empowering Graduate Engineering Students With Proficiency in Autonomy," in ASME 2018 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, 2018.
17. MathWorks. Obstacle Avoidance Using Adaptive Model Predictive Control. URL: https://www.mathworks.com/help/mpc/ug/obstacle-avoidance-using-adaptive-model-predictive-control.html.
18. J. Kong, M. Pfeiffer, G. Schildbach and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," in 2015 IEEE Intelligent Vehicles Symposium (IV), pp. 1094-1099, 2015.
19. Mathworks. Simulation Pacing. URL: https://www.mathworks.com/help/simulink/ug/simulation-pacing.html.
20. M. O'Kelly, V. Sukhil, H. Abbas, J. Harkins, et al., "F1/10: An Open-Source Autonomous Cyber-Physical Platform," arXiv preprint arXiv:1901.08567, no. 2019.
21. Y. Kuwata, G. A. Fiore, J. Teo, E. Frazzoli, et al., "Motion planning for urban driving using RRT," in 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1681-1686, 2008.
22. J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065), pp. 995-1001, 2000.

Contact Information

Ardashir Bulsara, 1239 Laurens View Road, Greenville SC, 29607.

abulsar@clemson.edu, +1(864)908-7252

Definitions/Abbreviations

ROS	Robot Operating System
MPC	Model Predictive Control
RST	Robotic Systems Toolbox
ACC	Adaptive Cruise Control
MoCap	Motion Capture System. In this case, this is the OptiTrack MoCap System.
CUICAR	Clemson University International Center for Automotive Research
RTW	Real Time Workshop/ Simulink Coder