Senior Honors Thesis

Robustifying Model Predictive Controllers using Scenic

Johnathan Chiu

Advisor(s): Edward Kim, Sanjit Seshia

March 2021

1 Introduction

Model Predictive Control (MPC) is generally used in scenarios where controls are needed for complex systems thus, very intricate, case-specific models need to be implemented to achieve high performance. It is often hard to create mathematical formulations of the surrounding conditions due to high stochasticity in the environment. Additionally, due to its need for high computational power, MPC controllers may fail to run in situations where the dynamics of the environment are complex mathematical equations which constantly and instantaneously change. With today's technology, MPC has proven to be a strong candidate to enable automation in a controls setting but, in many cases, some of the aforementioned dependencies of MPC can be costly and can often lead to disaster.

In greater detail, we find that an example of situations where stochasticity can lie is in modeling the effects of low temperatures and high wind speeds in an environment which expedite an icing process on the ground. This leads to low friction environments which are adversarial to any high-speed moving vehicle.

We attempt to produce stronger MPC controllers that perform with higher certainty[1] through empirical analysis and the use of our own modeling tool: SCENIC [1]. SCENIC provides a powerful framework that can generate a set of environments[2] which accurately model complex dependencies and stochasticity in the real world. In detail, at the beginning of each receding-horizon[3] of the system, we quickly and accurately compute a set of controls that we can execute to a degree of high certainty. Through each receding horizon, we generate a multitude of environment samples, taken from SCENIC, which accurately encapsulate potential changes of the environment within the receding horizon itself.

We demonstrate the aforementioned effects of sampling on MPC controllers through a system that executes automated airplane takeoff against adversarial

[1] We reference the term 'certainty' to mean both safety and accuracy

[2] We define environments to be any external dynamics imposed on the system

[3] In controls setting, this refers to the time elapsed before recomputing a new set of controls

wind speeds and directions. We make use of X-Plane flight simulator [3] to run experiments and evaluate performance of our new controllers. Using X-Plane's API, we can continuously change the wind conditions introducing high wind speeds and varying wind direction to compare a baseline MPC controller against our sampled controller. Using a predefined scoring system, we assess the abilities of the controller where a higher positive score demonstrates better performance.

2 Related Works

In previous literature, there have been many attempts to use the stochasticity of the environment to improve MPC. As seen in the article from Mesbah et. al [4], this involves trying to achieve higher robustness against the controller's uncertainties due to the receding-horizon formulation. One notable method involves robustifying MPC over worst-case scenarios. The main issue with this approach is that optimal actions are often overly conservative or infeasible. An approach similar to what we desire to achieve is that of *chance constraints*. In similar manner, we desire our system to be satisfied in expectation. We bound and provide deterministic descriptions of the environment surrounding our system. Despite the similarities in our approach, we allow our modeling tool, SCENIC, to handle the complex dependencies found in the environments around us.

3 MPC with Sampling

3.1 Sampling with SCENIC

3.1.1 SCENIC background

SCENIC is a probabilistic environment modeling language that allows users to specify distributions over environment semantic features. A few such use cases:

1. Specify distributions over behaviors of the environment (e.g. the wind magnitude every second over a Gaussian distribution with its mean equal to the current provided wind magnitude and estimated standard deviation).
2. Specify dependency relations between semantic features (e.g. when temperature is low and wind magnitude is high, the ground friction modeled as a joint Gaussian distribution).

Thus, a SCENIC program represents a distribution over the initial condition and expected behaviors and sampling from the program means selecting a specific instance of the environment variables from the given distribution. For further details, refer to SCENIC toolkit documentation [1].

3.1.2 Sampling

With SCENIC, we can explicitly define a set of parameters that model both wind speeds and wind headings we would find in real scenarios. We additionally provide the true measured wind speeds/heading and provide them as an input to SCENIC to be modeled as mentioned in section 3.2.1. By doing so, the solver should optimize over multiple scenarios in which the wind speeds and/or headings change instantaneously. This prevents the solver from outputting a set of controls over-fit to the current conditions measured at the beginning of the receding horizon.

3.2 Reformulation

3.2.1 Definitions

We define a set of variables denoted below:
f = model of system dynamics
g = associated cost function
$\vec{x}_t$ = states at time t
$\vec{u}_t$ = controls at time t
a_t = acceleration at time t
ω_t = angular velocity at time t
$\tilde{\omega}$ = physical angular velocity constraint of the plane
$\tilde{a}$ = physical acceleration constraint of the plane
E = true measured wind speed and heading
E_s = sampled wind speeds and headings (through SCENIC) over the receding horizon

3.2.2 Equations

At each iteration of MPC, we have an associated cost function denoted in equation (1). Our dynamics model, f, is the constraint over our plane and its physical states with respect to position, velocity, and acceleration in the x-axis and y-axis. One thing to note is that the value of $t+1$ is the time of the next receding horizon, not actual units of time in seconds.

$$\begin{aligned} \min_{x_{1:T}, u_{1:T}} \quad & \sum_{t=1}^{T} g(\vec{x}_t, \vec{u}_t, E) \\ \text{s.t.} \quad & \vec{x}_{t+1} = f(\vec{x}_t, \vec{u}_t, E) \quad \forall t \in \{1, ..., T\} \\ & |a_t| \leq \tilde{a} \quad \forall t \in \{1, ..., T\} \\ & |\omega_t| \leq \tilde{\omega} \quad \forall t \in \{1, ..., T\} \end{aligned} \tag{1}$$

Similarly, in equation (2), we propose a small modification to the cost – compute over the expected value of sampled environments from SCENIC. We consider the true, observed environment and our sampled environments equally and indistinguishably. The union function below implies that we simply add

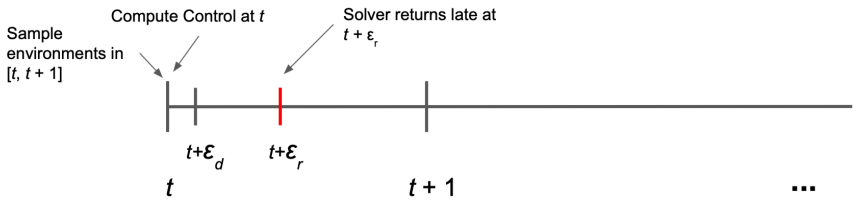

Figure 1: Issue with MPC solver. Note that ϵ_d is the desired maximum run-time for our solver and ϵ_r is the observed running time for the solver. In reality, when $\epsilon_r >> 0$, there are no controls applied over a long period of time.

the true environmental conditions to the set of sampled environments. Further, in computing the new cost function, we simply take individual cost values (and constraints) over each environment, e, and then average them to get our new cost.

$$\begin{aligned} \min_{x_{1:T}, u_{1:T}} \quad & \mathbb{E}_{e \in E \bigcup E_s}[\sum_{t=1}^{T} g(\vec{x_t}, \vec{u_t}, e)] \\ \text{s.t.} \quad & \vec{x}_{t+1} = f(\vec{x_t}, \vec{u_t}, e) \quad \forall t \in \{1, ..., T\} \\ & |a_t| \leq \tilde{a} \quad \forall t \in \{1, ..., T\} \\ & |\omega_t| \leq \tilde{\omega} \quad \forall t \in \{1, ..., T\} \end{aligned} \tag{2}$$

Intuitively, the solver will optimize over multiple scenarios thus creating robustness to instantaneous changes in our environment. We also assume the the environment is partially observable. This meaning that the time between which the environment is observed and actually computed over can change. As previously stated, we hope to use SCENIC to model these small changes.

3.3 Practical Uses

We find that computing the cost function presented in equation (2) takes too long to run in real-time. In MPC controllers, we desire near instantaneous solutions. With the current setup, computing the state update using our dynamics function over multiple environment samples takes longer than desired. Additionally, we find that the solver takes longer time to converge over the expected cost function. The issue is illustrated in further detail through Figure 1.

In section 3.2.2, we note that $t+1$ references the time following the receding horizon. Intuitively, the closer $t+1$ falls to t, the better the MPC controller performs. This implies that we are continuously updating our set of controls at a very fast rate.

3.3.1 Lookup Table

In situations where we need extremely quick computations, we can precompute a lookup table to determine a set of controls based off the current state in real-time. Although, using this method, there are certain negative implications in using a lookup table; a set of issues that arise are:

1. Lookup tables cannot be generalized and are specific to individual scenarios.
2. Lookup tables take a long time to generate and take up a lot of space in memory.
3. Lookup tables discretize the data over a continuous domain which can result in a build-up of errors.

Further, in initial experimentation with the lookup tables, we observed significant improvement in reducing overall cross track error(CTE) during take-off. However, testing with the lookup tables proved to be difficult given that re-implementing and tuning the cost function weights resulted in needing to regenerate the tables. Each table took over 24 hours to generate resulting in an incredibly slow experimentation process.

3.3.2 Kalman Filter

To enable our system to run in real-time, we implemented a Kalman Filter modeled using a linearized version of the dynamics in $f(\hat{x_t}, \hat{u_t}, E)$ which corresponds to the matrix $\boldsymbol{F}$ formally defined below. For any Kalman Filter, there exists two steps: a prediction and an update[4]. Using the given dynamics model, the Kalman Filter provides an initial prediction for the state of the system in transition. Afterwards, the update step corrects the predicted state by using a measurement of the system. We also must assume some uncertainty in the measurements and use an assumed covariance between individual elements of the state (e.g. covariance between heading and velocity). Finally, the minimization of error between the predicted and updated state works by applying a Kalman gain, $\boldsymbol{K}$, over the residual. Residual is the difference between a prediction and measurement. For a full derivation of the Kalman gain, see textbook from Kim et. al., 2019 [2].

Mathematically, we define our prediction step with two equations below. In equation 3, $\hat{x}$ is our prediction state vector, $\boldsymbol{F}$ is our dynamics model from equation, $\boldsymbol{B}$ is our controls matrix, and $\vec{u}$ is our control vector. In the following equation, the term $\boldsymbol{P}$ is the state error covariance and $\boldsymbol{Q}$ is some additive noise in the prediction. Intuitively, this means as more states are predicted, the uncertainty of our states should increase. We also use the superscript $+, -$ to denote posterior (updated) and prior (predicted) estimates as defined by Kim et. al., 2019 [2].

$$\begin{aligned} \hat{x}_k^- &= \boldsymbol{F}\hat{x}_{k-1}^+ + \boldsymbol{B}\vec{u}_{k-1} \\ \boldsymbol{P}_k^- &= \boldsymbol{F}\boldsymbol{P}_{k-1}^+\boldsymbol{F}^T + \boldsymbol{Q} \end{aligned} \tag{3}$$

[4]Prediction, in literature, is commonly referred to as "propogation" and update as "correction"

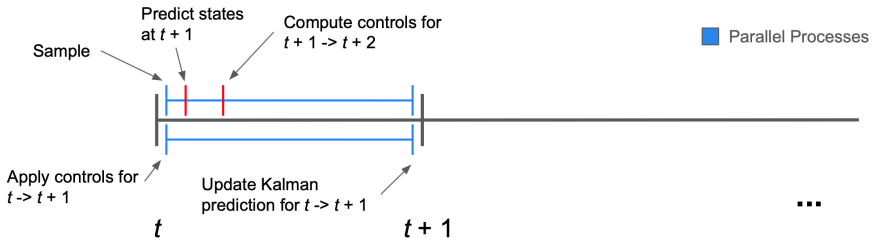

Figure 2: Parallel process: in the same receding horizon, our system both applies control from the predicted states and synthesizes control for the next horizon

Our update steps are defined by the set of equations in 4. We denote the variables $\tilde{y}$ to be our residual, z as our measurement vector, $\boldsymbol{H}$ as the measurement matrix, and $\boldsymbol{R}$ as the measurement noise. Additionally, $\boldsymbol{K}$ is the Kalman gain. At the end, we again need to update the error covariance.

$$\begin{aligned} \hat{y}_k &= z_k + \boldsymbol{H}\hat{x}_k^- \\ \boldsymbol{K}_k &= \boldsymbol{P}_k^- \boldsymbol{H}^T (\boldsymbol{R} + \boldsymbol{H}\boldsymbol{P}_k^- \boldsymbol{H}^T)^{-1} \\ \hat{x}_k^+ &= x_k^- + \boldsymbol{K}_k \hat{y}_k \\ \boldsymbol{P}_k^+ &= (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H})\boldsymbol{P}_k^- \end{aligned} \tag{4}$$

In using our Kalman Filter, we can predict states at the next receding horizon and feed those predictions into the solver during the current horizon in anticipation for the next horizon. This enables a parallel system where we apply the control instructions from the solver and compute upcoming controls for future states in parallel. We also couple our solver with a Proportional, Integrative, Derivative (*PID*) controller to apply the given set of controls at each time step. We demonstrate our solution in figure 2.

Evidently, our prediction of future states can only be so accurate and can lead to some inaccuracy for online computations. We discuss this more in the Conclusions section and allude to potential improvements with this system.

4 Experimentation

To test our hypothesis, i.e. robustifying MPC controllers with the use of SCENIC, we consider a series of test that run the controller with an increasing number of samples from SCENIC. Specifically, we consider sampling over 0, 5, 8, and 10 samples and the samples are given by SCENIC at the start of each receding horizon.

4.1 Setup

We desire that the plane can auto-taxi and take-off with certain constraints defined in our cost function. In specific, our cost function, $g(x_t, u_t, e)$, considers three elements: *heading error* (HE), *cross track error* (CTE), and terminal velocity. We do a weighted sum over all three elements that were tuned through experimentation.

We use a notable software for our cost function solver: scipy's optimization library [6]. We use its methods defined in the 'SLSQP' solver. In order to achieve the parallel computation described in section 3.3.2 we use Python's builtin multiprocessing library.

To run the simulations, we use X-Plane flight simulator from Laminar Research. Additionally, we use XPlaneConnect [5], developed by NASA, an API interface between the simulator and python to set conditions/environments and extract ground-truth data from the simulator.

In simulation, we vary the conditions within the the simulator every 3 seconds sampled over Gaussian distribution with mean defined by the observed values and a standard deviation of $1m/s$ for wind speed and $20°$ for wind heading. And, due to the limitations of our simulator, we found it difficult to work with other environment constraints. Like previously mentioned, we test over 4 controllers respectively with 0 sampling, 5 samples, 8 samples, and 10 samples.

4.2 Scoring

We give each controller 50 seconds to achieve terminal conditions for takeoff. If these conditions are not met, we allocate 40 more seconds to abort takeoff. We specified these time intervals through observing multiple simulation runs. Mathematically, we can further show, constraining the plane's acceleration to a low $a = 2m/s^2$, the plane should physically achieve far past the terminal velocity we desired. Using equation (5), and setting $v_0 = 0m/s, a = 2m/s^2, t = 50$ we get $v_f = 100m/s$. Finally, We run 300 simulations for each controller and evaluate its overall performance.

$$v_f = v_0 + at \tag{5}$$

Each simulation run is given a score defined over the controller's ability to 1. successfully takeoff or 2. safely abort takeoff if takeoff is not possible. We also consider the controller's ability to keep the plane as close to the center-line as possible. In table 1, we define the scoring system.

Elaborating on the terminology, the taxiing score is applied amongst all simulation runs and validates the controller's ability to keep near the center-line. "Takeoff Abort Failure" implies that the plane was not able to achieve terminal conditions and further, failed to abort the takeoff. "Takeoff Abort Success" is defined to be a plane unable to achieve terminal conditions but successfully aborted takeoff (a successful abort means that the plane goes to 0 velocity and stops near the center-line within the 40 given seconds). Finally, a "Successful Takeoff" means that the plane was able to achieve terminal velocity, a low *HE*, and a low *CTE*.

Experiment Scoring		
Score Type	Scoring Value	Specification
Center-line Violation	-10	Each time plane crosses $CTE > 3m$
Takeoff Abort Failure	-300	$v_f > 0m/s$, $CTE_f > 5m$
Takeoff Abort Success	-50	$v_f = 0m/s$, $CTE_f < 5m$
Takeoff Success	+100	$abs(HE_f) < 2°$, $v_f > v_{terminal}$, $CTE_f < 3m$

Table 1: Scoring table to classify each simulation run.

4.3 Results

Generally speaking, sampling gives us better performance as shown by the trajectory graph and scores in the provided figure 3 and table 2, respectively.

Simulation Results		
Number of samples	Average Score over 300 runs	Takeoff Success Rate
0	-203.33	30.2%
5	10.52	70.8%
8	18.13	72%
10	-114.67	46.7%

Table 2: Results of experiments.

Interestingly, there is a notable improvement with 10 samples with respect to *CTE*. However, it fails to satisfy terminal velocity. Our intuition is that trying to synthesize a control that works in various environments result in conservatism. In this case we observe this as inability to satisfy the terminal velocity constraints. However, we may be able to address this tradeoff by giving more weights to the terminal velocity constraint. We hope to capture this trade-off between the number of sampled environment and the conservatism we see in future works.

5 Conclusion

In this paper, we demonstrate the use of SCENIC and its ability in improving Model Predictive Control. In specific, we demonstrated:

1. Ability to robustify MPC methodology through different environmental conditions.
2. Show how to simplify dynamics models that can be used for similar/overlapping uses.
3. Introduce modifications to the feedback loop that allow for online computation of the MPC controller.

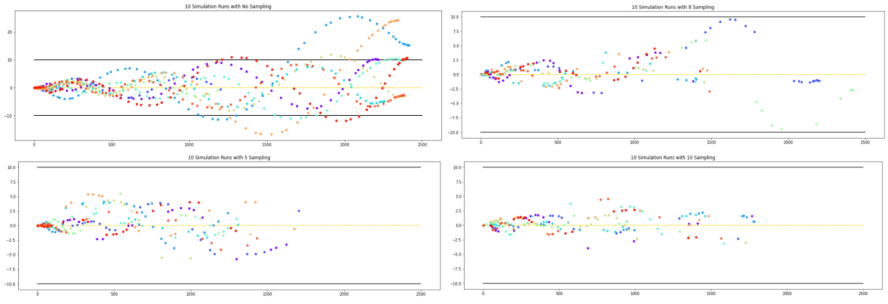

Figure 3: 10 runs extracted from simulation. [upper left: 0 sampling, upper right: 8 sampling, bottom left: 5 sampling, bottom right: 10 sampling]

We additionally introduce a novel solution for enabling online computation for MPC coupled with SCENIC. In these set of experiments, we only consider two two independent environment parameters due to the limitation of the simulator. Though this is a simple demonstration, with Scenic, developers can easily model stochastic and dependent relations in the complex environment and use this model to generate samples of feasible future environments based on current observed environment. Through this, synthesize a robust control that minimizes the cost over the sampled environments.

While there is a clear improvement from sampling environments, we believe that the accuracy of the system can be greatly refined with a better Kalman Filter. We observed higher accuracy in the use of our lookup tables (though not shown in this paper). Since we used a simple Kalman Filter, an unscented/extended Kalman Filter could enhance the accuracy of the prediction given our system is not linear. We can also reduce the number of predicted steps and find a explore trade-offs between filter accuracy and solver speed. Additionally, we could similarly predict the states over all sampled environments and average them rather than taking the prediction over only the truth environment conditions.

We hope to further investigate the results seen in the controller with 10 samples by testing with potentially greater number of samples. However, we are certain that increasing samples may lead to computational issues in which the time to compute will definitely exceed the receding-horizon. Given this, we will attempt to value the trade-off between number of samples and the conservatism within the solver. As stated previously, we also may need to look into reformulating the problem itself to achieve better performance for a larger number of samples.

Finally, incorporating the final computed cost could greatly improve the number of successful takeoffs. The solver can see the trajectory over all states and determine the achievability of terminal conditions. Intuitively, a high cost function with a singular good final state and immediate takeoff is a poor decision

takeoff (i.e. our plane oscillates over the centerline but momentarily achieves terminal conditions). Whereas if the cost function remains low, the plane must be following the centerline, maintaining correct heading, and exceeding terminal velocity over a number of states.

References

[1] Daniel Fremont et al. *Scenic: Language-Based Scene Generation*. Tech. rep. UCB/EECS-2018-8. EECS Department, University of California, Berkeley, Apr. 2018. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-8.html.

[2] Youngjoo Kim and Hyochoong Bang. "Introduction to Kalman Filter and Its Applications". In: *Introduction and Implementations of the Kalman Filter*. Ed. by Felix Govaers. Rijeka: IntechOpen, 2019. Chap. 2. DOI: 10.5772/intechopen.80600. URL: https://doi.org/10.5772/intechopen.80600.

[3] Laminar Research. *X-Plane 11*. 2019. URL: https://www.x-plane.com/.

[4] Ali Mesbah. "Stochastic Model Predictive Control: An Overview and Perspectives for Future Research". In: *IEEE Control Systems Magazine* 36.6 (2016), pp. 30–44. DOI: 10.1109/MCS.2016.2602087.

[5] Christopher Teubert and Jason Watkins. *The X-Plane Connect Toolbox*. 2019. URL: https://github.com/nasa/XPlaneConnect.

[6] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.