Model Predictive Control in Deformable Terrain

Alex Pletta
MS Robotic Systems Development
Carnegie Mellon University
Pittsburgh, USA
apletta@andrew.cmu.edu

Benjamin Younes
MS Robotic Systems Development
Carnegie Mellon University
Pittsburgh, USA
byounes@andrew.cmu.edu

Russell Schwartz
MS Robotics
Carnegie Mellon University
Pittsburgh, USA
rwschwartz@cmu.edu

Jacob Wang
Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, USA
zhuoyuaw@andrew.cmu.edu

Changliu Liu
Assistant Professor
Carnegie Mellon University
Pittsburgh, USA
cliu6@andrew.cmu.edu

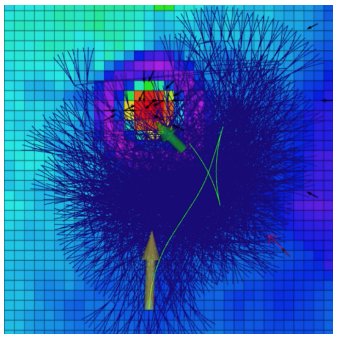
***Abstract*—In this project, we consider the problem of path planning and control for an autonomous sitework robot in deformable terrain. We aim to drive the robot from any initial pose to any desired goal pose with optimal path and minimal control. We split the task into two subtasks: reference path generation and model predictive control (MPC). In the first phase, we use Dubin's path planning and simple stabilizing control to generate reference trajectories from the initial pose to the goal pose. In the second phase, we derive a reference tracking MPC to find actual controls for the system that satisfies physical constraints. We then combine the two and numerically simulate the closed-loop controller performance in presence of steering actuator lag and dynamic slip. Results for the different reference trajectory approaches and slip impact are discussed and show the Dubin's MPC method is effectively able to drive the system between arbitrary poses while safely minimizing the control input within constraint bounds.**

***Index Terms*—model predictive control, automated driving, deformable terrain, trajectory optimization**

I. Introduction

Students at Carnegie Mellon University are working to return to the lunar surface with a robotic presence by the end of 2023 with many missions slated for the next five years. One interest in robot lunar systems is the ability to manipulate terrain in order to build early infrastructure such as landing pads, roads, or structural foundations on the surface of the moon. CMU, Masters in Robotic System Development (MRSD), students built an autonomous research robotic platform to manipulate lunar regolith to facilitate this lunar surface robotics research. Functionally, the mobility planning and control aspect of the robot is required drive to a set of given goal poses from its current pose within a sandbox (Figure 1). Position and heading thresholds determine when a goal pose has been reached and if re-planning is needed.

The existing trajectory planning solution is an A* kinematic lattice planner (Figure 1). This approach produces feasible trajectories but does not consider dynamic effects, can take time to compute, is challenge to tune robustly, and in some cases needs to sacrifice path optimality to find a solution.



(a) Kinematic lattice planner path to crater goal pose.

(b) Robot in testing sandbox.

Fig. 1. Autonomous sitework robot system.

The existing solution for control is decoupled steer and drive controllers. The steer controller uses a weighted sum of cross track error and trajectory steering angle. The drive controller scales a maximum speed down depending on steering speed. This ad hoc approach is challenging to tune and results in conservative performance.

This work seeks to improve the planning and control pipeline of this robot via a Model Predictive Control solution. The solution will generate both a trajectory with drive and steer control sequence.

An initial goal of the project was to implement the full MPC pipeline on the robot hardware, but this has been descoped to pure simulation. Although the project is not running on hardware, the intent of the work described was to replicate the constraints and feasibility as if it were running on the intended robot.

II. Literature Review

The survey paper by Snider [1] is an outline for comparison between planners and controllers, and outlines some LQR work for control.

Model predictive control (MPC) is widely used for control of automobiles [2], robot arms [3], humanoid robots [4] and *etc* [5], [6]. Besides the standard MPC shown in the lecture, there are many variants of it for different scenarios and purposes. For example, MPC with iterative linearization [7] deals with nonlinear dynamics by linearizing at each operating point over the MPC outlook horizon. Nonlinear MPC [8] considers full nonlinear dynamics of the system for dynamics constraints, while the online computation could be an issue. More recently, deep neural network is used to learn the latent features in the problem and combined with MPC to form DeepMPC [9].

III. Methodology

The goal poses are generated through abstracted planners upstream, and the current agent state is given by a localization system, both of which are assumed to be implemented and tested prior to this work.

In this project we develop an implementation of Model Predictive Control for trajectory and control sequence generation. The controller first generates a reference trajectory using a simplified system model. We then use MPC to track the reference trajectory using the full system dynamics while respecting the constraints on vehicle kinematics (e.g. minimum steering radius, turning acceleration, max turn rate), control (e.g. steering, drive), and state (e.g. position, speed). This implementation allows for extensions and augmentations depending on initial performance.

Using a model of the robot dynamics, the optimal future control sequence for a fixed (finite) time horizon will be computed at each time step by solving a quadratic program with respect to the reference trajectory. The first step in this sequence will be executed, and the then the system will re-plan. The full-fidelity robot dynamics are described in section IV.

We explore two methods for generating reference trajectories: a unicycle-model-based stabilizing controller, and controller that uses Dubin's path planning algorithm. These approaches are discussed in section V.

IV. System State and Dynamics

The robotic platform is built upon a heavily modified RC car platform which has a ackermann steering module in the front and back of the vehicle. These ackermann steering modules are connected mechanically in such a way that they achieve a roll average suspension.

We use a double kinematic bicycle model, in which the vehicle turns in perfectly circular arcs whose radius is determined by a wheel steering angle. The state x consists of position (p_x, p_y), heading angle θ, and steering angle ψ. The control consists of forward velocity v and the change in steering angle $\dot{\psi}$.

$$x = \begin{bmatrix} p_x \\ p_y \\ \theta \\ \psi \end{bmatrix} \qquad u = \begin{bmatrix} v \\ \dot{\psi} \end{bmatrix}$$

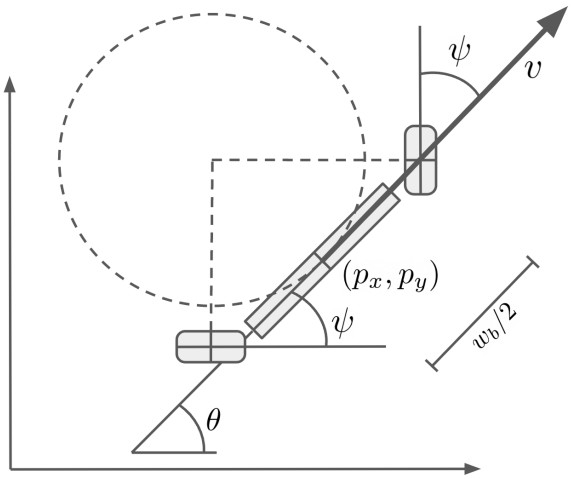


Fig. 2. Double Kinematic Bicycle Model. The quantity w_b represents the distance between the steering pivots of the front and rear wheels.

The discrete time dynamics are as follows:

$$x_{t+1} = \begin{bmatrix} p_{xt} \\ p_{yt} \\ \theta_t \\ \psi_t \end{bmatrix} + \begin{bmatrix} v_t \cos(\psi_t)\cos(\theta_t) \\ v_t \cos(\psi_t)\sin(\theta_t) \\ v_t \sin(\psi_t)\frac{w_b}{2} \\ \dot{\psi}_t \end{bmatrix} \delta_t \tag{1}$$

We impose the following constraints on the state and control:

- steer angle limit: $|\psi| \leq 0.5$ (radians)
- steer angle velocity limit: $|\dot{\psi}| \leq 1.0$ (radians/s)
- velocity limit: $|v| \leq 1.0$ (m/s)

We also attempted to model slip, using a naive approach, as shown below,

$$x_{t+1} = \begin{bmatrix} p_{x_{t+1}} \\ p_{y_{t+1}} \\ \theta_{t+1} \\ \psi_{t+1} \end{bmatrix} = \begin{bmatrix} p_{x_t} \\ p_{y_t} \\ \theta_t \\ \psi_t \end{bmatrix} + \begin{bmatrix} v_t\delta_t(cos(\psi_t)\cos(\theta_t) + S_t^{(0)}k_x) \\ v_t\delta_t(cos(\psi_t)\sin(\theta_t) + S_t^{(1)}k_y) \\ v_t\delta_t(sin(\psi_t)\frac{w_b}{2} + S_t^{(2)}k_\theta) \\ \dot{\psi}_t\delta_t + S_t^{(3)}k_\psi \end{bmatrix}$$

This non-additive noise uses some gains which are hand crafted in order to inject process noise into the state. This was tuned by looking at the testing data from the vehicle running in the sandbox, but was not done rigorously. The 4×1 random vector, S_t was drawn from a uniform distribution $S_t \sim U(-1, 1)$, every N seconds. This slip was modeled in the global frame and not the vehicle frame as a design decision, the accuracy of which could be improved upon greatly if more time was devoted to characterising slip. In its current manifestation, the slip model serves to introduce non-additive, non-Gaussian process noise to help test the robustness of the MPC solver.

V. Reference Trajectories

We explore two different methods for generating a semi-feasibly reference trajectory.

A. Heading-Agnostic Stabilizing Control

In the case of a unicycle model with control on $\dot{\theta}$, we can derive a stabilizing control law from the Lyapunov function:

$$v = -\frac{k_v}{dt}\left(p_x \cos\theta + p_y \sin\theta\right)$$
$$\dot{\psi} = -\frac{k_\theta}{dt}\theta$$

This control law will stabilize the system to the origin, without regard to the desired goal heading. We adapt this solution to our control scheme as follows. Since we cannot control $\dot{\theta}$ directly, we treat ψ as a proxy for $\dot{\theta}$. We then run a first-order controller on top of $\dot{\psi}$. We generate the reference trajectory by running this closed loops controller and recording the state history. Finally, we modify the last few reference states to match the goal heading exactly at time $t = t_{\max}$.

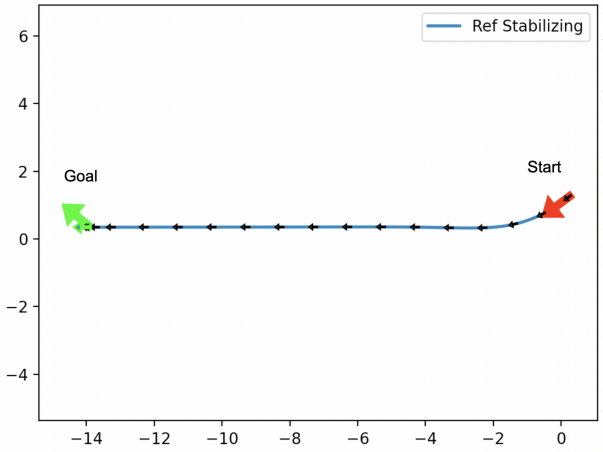


Fig. 3. Example reference trajectory generated with stabilizing controller.

B. Dubin's Path Planning

A *Dubin's path* is a geometrically optimal solution to the problem of finding a curve of minimum length that connects two given points with given tangent vectors under a constraint on curvature [10]. This is a good reference problem for our system since it allows us to account for both the curvature turning-radius constraint and the terminal heading constraint (unlike the unicycle stabilizing controller).

Intuitively, a Dubin's path always consists entirely of straight line segments and circular arcs. Moreover, these arcs always exploit the maximum allowed curvature. One drawback of this solution in our context is that it does not allow for the vehicle to move backwards, which in reality our rover can do without issue (e.g. to execute a K turn). A generalization of Dubin's paths called *Reeds–Shepp curves* allow for vehicle reversing [11]. Implementing Reeds-Shepp curves as a potential reference trajectory is an avenue for potential further work.

Dubin's path gives an optimal reference trajectory for the MPC to follow. As shown above in Figure 4, Dubin's path allows for a deviation from the shortest Euclidean path in order

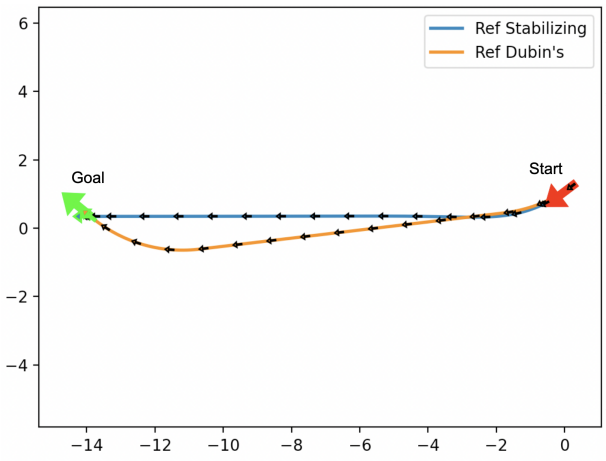


Fig. 4. Example reference trajectory generated with stabilizing vs. Dubin's path.

to arrive at the goal with desired heading. For better tracking ability and easier tuning, Dubin's minimum turning radius was inflated to allow for the vehicle to track with authority when slip pushes it off-path.

VI. Model Predictive Control

For simplicity and computation efficiency, in this project we use model predictive control (MPC) with linearized system dynamics. Specifically, at each time step k, we linearize the system dynamics (1) at x_k. Assume that we have the reference state and control trajectories x_{ref} and u_{ref} from section V. We solve the following optimization

$$\begin{aligned} \text{minimize}_z \quad & \frac{1}{2} z^T P z + q^T z \\ \text{subject to} \quad & Cz = d \\ & Gz \leq h \end{aligned} \tag{2}$$

where $z = [u_k, x_{k+1}, u_{k+1}, \cdots, u_{k+T-1}, x_{k+T}]$ is the concatenated controls and states with T being the MPC outlook horizon. The equality constraint $Cz = d$ is the dynamics constraint, where

$$C = \begin{bmatrix} B & -I & & & & \\ & A & B & -I & & \\ & & & & \ddots & \\ & & & & A & B - I \end{bmatrix}$$

with A and B being the linearized system dynamic matrices and $d = \mathbf{0}$. The inequality constraint $Gz \leq h$ is the state and control constraints listed in section IV, where

$$G = \begin{bmatrix} 0 & I & & & & & & \\ 0 & I & & & & & & \\ & & 0 & I & & & & \\ & & 0 & -I & & & & \\ & & & & \ddots & & & \\ & & & & & 0 & I & 0 \\ & & & & & 0 & -I & 0 \end{bmatrix}$$

and $h = [0.5, 1.0, 1.0, \cdots, 0.5, 1.0, 1.0]^T$. Matrices P and q are given by

$$P = \begin{bmatrix} R & & & & \\ & Q & & & \\ & & \ddots & & \\ & & & R & \\ & & & & Q_f \end{bmatrix}, \quad q = \begin{bmatrix} -R(\bar{u}_1 - u_{eq}) \\ -Q(\bar{x}_2 - x_{eq}) \\ \vdots \\ -R(\bar{u}_{T-1} - u_{eq}) \\ -Q_f(\bar{x}_T - x_{eq}) \end{bmatrix}$$

Here Q, R, Q_f are state cost, control cost and terminal state cost matrices, $\bar{x}$ and $\bar{u}$ are the reference state and control trajectories, x_{eq} and u_{eq} are the equilibrium points that we linearize the dynamics on.

Note that since C only contains linearized system dynamics, the constraints in the optimization are all linear. And we form the objective of the optimization being quadratic with regard to the variables. Thus optimization (2) is a quadratic program (QP), which can be solved efficiently online with many existing solvers.

The overall MPC procedure is shown below.

Algorithm 1 Model Predictive Controller

Initialize x_0
$k \leftarrow 0$
Generate reference trajectories $\bar{x}$ and $\bar{u}$
while $k < N$ **do**
  Observe x_k
  Linearize system dynamics at x_k and get A and B
  Formulate the MPC optimization (2) with A and B
  Find $u_k \leftarrow$ **solve** $\{z$ in (2)$\}$
  Execute action u_k
  $k \leftarrow k + 1$ on the actual system dynamics
end while

Furthermore, in order to better capture the nonlinear dynamics of the system, we iteratively linearize the system dynamics along the MPC trajectory to feed back into the MPC optimization. Specifically, we generate the MPC control and state trajectory as in Algorithm 1, and then compute the linearized dynamics matrices A_i and B_i around state and control x_i and u_i for all $i \in \{0, 1, \cdots T-1\}$. We use such A_i and B_i to again construct the equality constraint in the MPC, where we have

$$C = \begin{bmatrix} B_0 & -I & & & & & \\ & A_1 & B_1 & -I & & & \\ & & & & \ddots & & \\ & & & & & A_{T-1} & B_{T-1} - I \end{bmatrix}$$

and

$$q = \begin{bmatrix} -R(\bar{u}_1 - u_{eq}^{(1)}) \\ -Q(\bar{x}_2 - x_{eq}^{(2)}) \\ \vdots \\ -R(\bar{u}_{T-1} - u_{eq}^{(T-1)}) \\ -Q_f(\bar{x}_T - x_{eq}^{(T)}) \end{bmatrix}$$

where $x_{eq}^{(i)}$ and $u_{eq}^{(i)}$ are the states and controls we linearize our dynamics on at step i.

The overall procedures for this new MPC with iterative linearization is shown below in Algorithm 2.

Algorithm 2 MPC with iterative linearization

Initialize x_0
$k \leftarrow 0$
Generate reference trajectories $\bar{x}$ and $\bar{u}$
while $k < N$ **do**
  Observe x_k
  Linearize system dynamics at x_k and get A and B
  Get MPC state and control trajectories $[u_0, x_1, u_1, \cdots, u_{T-1}, x_T]^T$ from the MPC optimization (2) with A and B
  Linearize the system dynamics along the MPC trajectory $[x_i, u_i]$ to get A_i and B_i for all $i \in \{0, 1, \cdots T-1\}$
  Formulate the new MPC optimization (2) with A_i and B_i
  Find u_k from the new MPC optimization
  Execute action u_k
  $k \leftarrow k + 1$ on the actual system dynamics
end while

While iterative linearization was implemented and tested, it did not have a beneficial impact on the resulting MPC solutions and in some cases performed worse than the linearization at x_k. Further refinement and tuning of the iterative linearization may yield better performance, but for now all results are reported for MPC formulation with linearization about x_k.

VII. Results

The reference controllers and MPC solutions were evaluated against randomized starting and ending goal poses. The starting and ending poses were constrained by positional and heading difference to ensure a feasible solution was possible. Path trajectory, control, and state tracking results were collected for both MPC solutions with respective reference controllers for one trial without the slip model and one trial with the slip model. The same start and end poses were used for both trials to simplify comparability. Algorithms terminated early if within $0.1\mathtt{m}$ and $0.05\mathtt{rad}$ of the goal position and heading, respectively, and otherwise allowed to run until reaching a maximum number of iterations; for this report clamped to 401. Both MPC results used the linearization about x_k instead of the iterative linearization. Analysis of these results is contained in Section VIII.

Figures 5 and 6 compare the simulated path trajectories for both slip trials. Figures 7 and 8 compare the computed control commands while Figures 9 and 10 compare the individual state tracking performance, again for all controllers and both no-slip vs. slip trials.

Metrics in this report include the error between trajectory end and desired goal for all states, number of algorithm iterations for solution, and summed derivatives for velocity and change in steering angle as an approximation for energy

requested. Trajectory error is reported in Table I and the algorithm iterations with energy requested are reported in Table II.

The terminal trajectory pose error is included to gauge how well each algorithm was able to reach the goal, regardless of runtime performance. The number of algorithm iterations is included as a proxy for algorithm speed and complexity; in general algorithms may spend more time per iteration depending on their complexity, but this metric is still useful for estimating holistic solution speed. Finally, the summed derivatives of velocity and change in steering input are included to estimate the effective energy requested by each control trajectory. The derivative of the velocity command can be considered as drive acceleration, and the derivative of the change in steer command can be considered as steering angle acceleration. Both accelerations would require a certain amount of work done by the actual system, so are summed to capture how much energy the control sequence would theoretically use. Note that these values would change depending on real system power efficiency, so only the relative magnitudes are important for the sake of comparing the algorithms in this report.

All results were generated using the following Python code base, which are linked here.

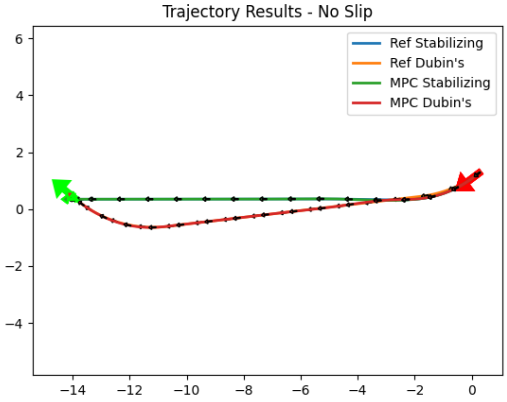


Fig. 5. Final path trajectories without slip. Trajectory arrows are placed at 1 second travel intervals aligned with heading direction. The large red and green arrows indicate start and end poses, respectively. Axes in meters.

TABLE I. Control trajectory error between trajectory end and goal pose. Δx_f and Δy_f in meters, $\Delta\theta_f$ and $\Delta\psi_f$ in radians. NS and S indicate No-Slip and Slip trials, respectively.

Method	Δx_f	Δy_f	$\Delta\theta_f$	$\Delta\psi_f$
Ref. Stabilizing (NS/S)	**0.010**	**0.000**	0.688	**0.000**
Ref. Dubin's (NS/S)	0.126	-0.110	**0.014**	-0.500
MPC Stabilizing (NS)	**0.005**	**0.000**	0.688	**0.000**
MPC Dubin's (NS)	0.059	-0.058	**-0.004**	-0.447
MPC Stabilizing (S)	**0.004**	**-0.014**	0.672	**-0.003**
MPC Dubin's (S)	0.015	-0.040	**-0.037**	-0.463

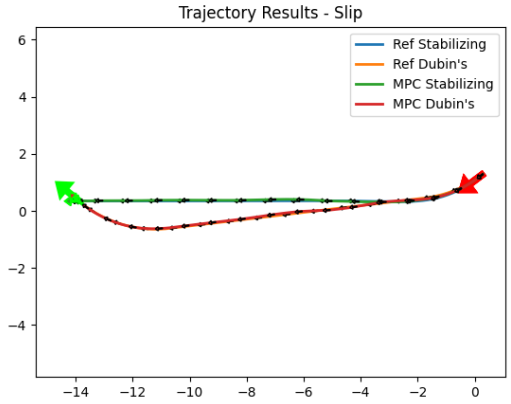


Fig. 6. Final path trajectories with dynamic slip model. Trajectory arrows are placed at 1 second travel intervals aligned with heading direction. The large red and green arrows indicate start and end poses, respectively. Axes in meters.

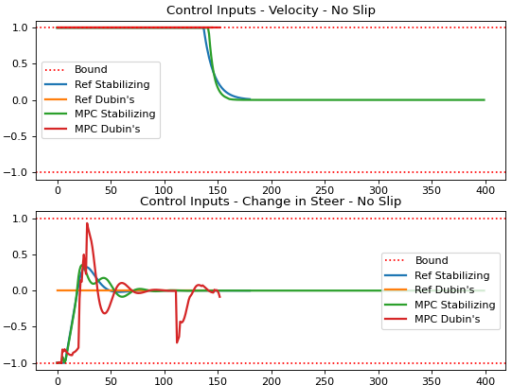


Fig. 7. Final control input trajectories without slip. Control bounds marked by red dashed lines. Velocity and Change in Steer reported in units of $\frac{m}{s}$ and $\frac{rad}{s}$, respectively, over time step.

VIII. Discussion

A. Results Analysis

The algorithm performances in this report yield several interesting results through comparisons between the stabilizing and Dubin's approaches combined with the no-slip vs. slip trials.

1) Tracking Performance: In general, the stabilizing controller and resulting MPC solution tracked efficiently to the target goal but did not correct for final heading while Dubin's planner and the associated MPC solution did correct for final heading by deviating more from the Euclidean-shortest path to the goal. The stabilizing controller solution did not terminate with threshold of the goal pose while Dubin's MPC did. With the amount of slip simulated, minor MPC tracking deviations

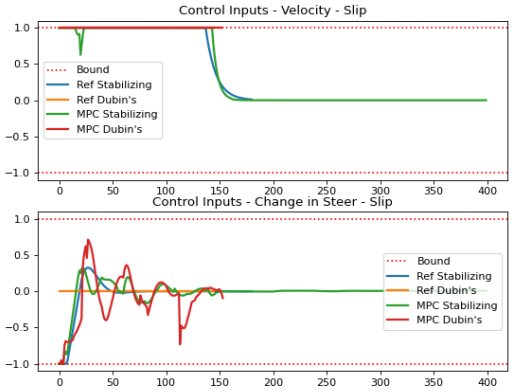


Fig. 8. Final control input trajectories with dynamic slip model. Control bounds marked by red dashed lines. Velocity and Change in Steer reported in units of $\frac{m}{s}$ and $\frac{rad}{s}$, respectively, over time step.

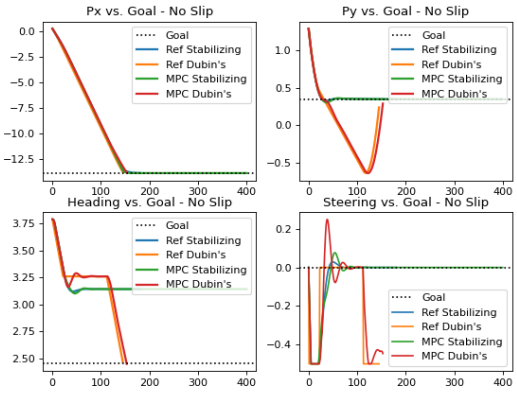


Fig. 9. State tracking results without slip. State goals marked by black dashed lines. Positions (Px, Py) reported in meters and angles (Heading, Steering) reported in radians, both over time step.

from the reference path can be observed as illustrated in Figures 5 and 6 for no-slip and slip, respectively. As shown in Table I, the terminal stabilizing solution was closer to the goal pose for position and steering angle, but with much more heading error than the Dubin's solution. This error is also visualized in Figures 9 and 10, where the Dubin's solution can be observed to increase and then decrease more Py error to allow for better heading while the stabilizing solution maintains large heading error until termination. The state tracking with slip is slightly nosier than with no-slip, though in both trials the controllers were stable and moved the system towards the final goal states.

2) Velocity Control: Both the Dubin's MPC associated reference control maximize velocity, while the stabilizing solution guides velocity to zero towards the end of the trajectory as

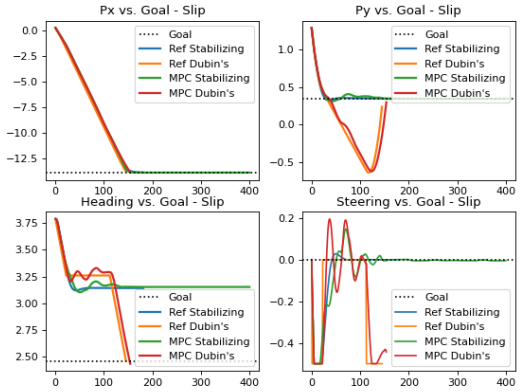


Fig. 10. State tracking results with dynamic slip model. State goals marked by black dashed lines. Positions (Px, Py) reported in meters and angles (Heading, Steering) reported in radians, both over time step.

TABLE II. Holistic trajectory solution performance. Algorithms terminated early if within $0.1m$ and $0.05rad$ of the goal position and heading, respectively, and ran until the maximum number of iterations. The max iterations was clamped to 401. Energy for velocity E_v and steering control E_ψ in units of $\frac{m}{s^2}$ and $\frac{rad}{s^2}$, respectively, as the summation of differentiated control inputs v and $\dot{\psi}$. NS and S indicate No-Slip and Slip trials, respectively.

Method	Iterations	E_v	E_ψ
Ref. Stabilizing (NS/S)	182	9.9	16.9
Ref. Dubin's (NS/S)	146*	0.0	0.0
MPC Stabilizing (NS)	401	10.0	**22.8**
MPC Dubin's (NS)	154	**0.0**	64.3
MPC Stabilizing (S)	401	49.2	**58.5**
MPC Dubin's (S)	156	**0.0**	74.1

*No iteration in Dubin's algorithm; iterations reported as number of trajectory poses for approximated complexity comparison.

seen in Figures 7 and 8. This result is due to how the reference control trajectories were formed and because the final goal state does not have velocity at all, so terminating with non-zero velocity is a valid result in this case. The stabilizing reference controller is designed to reduce velocity as the goal is reached, which the MPC solution tracks effectively. In contrast, Dubin's planner computes a state trajectory assuming a single constant velocity. The Dubin's reference control trajectory was then crafted as only the maximum velocity with zero input to change in steer to help guide the Dubin's MPC to drive as fast as safely possible while minimizing steering change. However, without a constraint on terminal velocity the MPC solution does not reduce speed prior to reaching the goal. The overall velocity control trajectories for both algorithms as compared between slip trials are roughly the same, but the stabilizing MPC has a sharp drop in velocity while the Dubin's MPC maintains a constant velocity.

3) Steering Control: The calculated change in steering control inputs for the stabilizing solution are slightly smoother than for the Dubin's solution, as can be observed in Figures 7 and 8. This result is most likely because the stabilizing solution does not turn as much to correct for final heading error. Both controllers start with maximum steering commands that quickly move back towards the other direction to correct for slight overshoot in Py, shown in Figures 9 and 10, to guide the motion towards the goal. The stabilizing controller relatively quickly aligns with the Euclidean-shortest path to the goal and then does not change steering for the remainder of the trajectory. In contrast, the Dubin's MPC controller has larger amplitude of steering inputs to align the motion along a straight line path before having a large negative steering input to turn the motion to the right and align with the goal heading. Again, the Dubin's reference controller is crafted with zero change in steer to encourage the MPC solution to limit steering change while still allowing the MPC to find a feasible solution. Both control approaches have similar profiles with more noise in the slip trial than for the no-slip trial in order to account for the unexpected state changes due to the slip.

4) Approximate Control Energy Consumption: The approximate complexity for the reference stabilizing controller and the Dubin's planning controller were roughly the same; both algorithms solved in similar amounts of time with similar amount of trajectory waypoints. However, the stabilizing MPC consistently ran for the maximum number of iterations while the Dubin's MPC often found a solution within threshold to the goal for early termination. The number of iterations used by each algorithm are shown in The approximate complexity for the reference stabilizing controller and the Dubin's planning controller were roughly the same; both algorithms solved in similar amounts of time with similar amount of trajectory waypoints. However, the stabilizing MPC consistently ran for the maximum number of iterations while the Dubin's MPC often found a solution within threshold to the goal for early termination. Table II reports these algorithm iterations, along with the associated energy used to control velocity and change in steering angle for each algorithm. The stabilizing MPC used more energy for velocity because it decreased velocity towards the end of the trajectory. Dubin's MPC technically used zero energy for velocity control because it commanded a constant velocity for the full trajectory, though constraining terminal velocity to be zero would likely make this controller calculate a similar velocity trajectory, and resulting energy consumption, to the stabilizing MPC solution. The stabilzing MPC used less energy for change in steering, though this is again because the algorithm approach did not correct for heading. Dubin's MPC used more steering energy in order to turn back towards the goal pose in order to arrive at the correct goal heading, as shown in Figures 5 and 6. Both controllers used more energy for the slip trial than the no-slip trial, as would be expected in order to account for path deviation.

B. Lessons Learned

As the team as implemented it, MPC seems oddly brittle and difficult to tune. There were cases where adding too much process noise made the solution much worse qualitatively then the stabilizing reference controller. In order to implement MPC in a robust and fault tolerant manner, the team now understands that more layers need to be added to raw MPC. We also now have a greater respect for MPC as a tool within a larger planning and control context. The design trade off between constraining MPC and giving it more authority to deviate from a near optimal reference trajectory is an interesting case study that requires more time. Overall, as a group we feel much more grounded to implementing an optimal control approach with non-linear dynamics.

C. Future Work

Adding the full Reeds-Shepp curves for planning would allow for the vehicle to reverse, a critical function for all autonomous mobile robots. Implementing this from scratch becomes non trivial due to the number of geometric cases that need considering, but would be an excellent step forward if more time was allotted. Re-tuning the iterative linearization may be useful for calculating the lifted dynamics for longer trajectories. Testing the algorithm to unknowns in state or control may be useful, as we only investigated a perfect observation model. Another approach not explored in this planning approach would be to try adding Dubin's and A* together in order to incorporate topography cost for terrain relative navigation and hazard avoidance.

Finally, once things are stabilized on reference trajectory and control, the final step would be to implement this stack using the live vehicle in the relevant environment, with the unprecise control and localization. This would require lots of time to tune and debug as it would have to be ported to C++ in order to interface with the ROS2 stack, but would be a great way to test our implementation in the real world to validate the algorithms and modeling approaches.

IX. Conclusion

In this project, we analyzed both stabilizing and Dubin's planning reference trajectories for MPC control of a simulated autonomous sitework robot to arbitrary goal poses. Non-linear motion dynamics were modeled with steering lag and dynamic slip. The reference trajectory generation, stabilizing control, Dubin's planning algorithms, and the linearized MPC implementations were supported by course theory, reference texts, and professor feedback. The trajectory tracking and control input performance for each controller are visualized and quantified relative to each other. Numerical simulation demonstrates successful tracking performance for Dubin's MPC even with the dynamic slip model. In future work we highlight subsystem improvements to run on real robot hardware. As a team, we feel holistically more capable of implementing optimal non-linear control on real world robots and understand process to go from the design of a state and control to a fully working system.

REFERENCES

[1] J. M. Snider *et al.*, "Automatic steering methods for autonomous automobile path tracking," *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RITR-09-08*, 2009.

[2] L. Del Re, F. Allgöwer, L. Glielmo, C. Guardiola, and I. Kolmanovsky, *Automotive model predictive control: models, methods and applications*. Springer, 2010, vol. 402.

[3] E.-H. Guechi, S. Bouzoualegh, L. Messikh, and S. Blažic, "Model predictive control of a two-link robot arm," in *2018 International Conference on Advanced Systems and Electric Technologies (IC_ASET)*. IEEE, 2018, pp. 409–414.

[4] T. Erez, K. Lowrey, Y. Tassa, V. Kumar, S. Kolev, and E. Todorov, "An integrated system for real-time model predictive control of humanoid robots," in *2013 13th IEEE-RAS International conference on humanoid robots (Humanoids)*. IEEE, 2013, pp. 292–299.

[5] D. H. Shim, H. J. Kim, and S. Sastry, "Decentralized nonlinear model predictive control of multiple flying robots," in *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*, vol. 4. IEEE, 2003, pp. 3621–3626.

[6] M. Wang, J. Luo, and U. Walter, "A non-linear model predictive controller with obstacle avoidance for a space robot," *Advances in Space Research*, vol. 57, no. 8, pp. 1737–1746, 2016.

[7] A. Carvalho, Y. Gao, A. Gray, H. E. Tseng, and F. Borrelli, "Predictive control of an autonomous ground vehicle using an iterative linearization approach," in *16th International IEEE conference on intelligent transportation systems (ITSC 2013)*. IEEE, 2013, pp. 2335–2340.

[8] F. Allgöwer and A. Zheng, *Nonlinear model predictive control*. Birkhäuser, 2012, vol. 26.

[9] I. Lenz, R. A. Knepper, and A. Saxena, "Deepmpc: Learning deep latent features for model predictive control." in *Robotics: Science and Systems*, vol. 10. Rome, Italy, 2015.

[10] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, no. 3, pp. 497–516, 1957. [Online]. Available: http://www.jstor.org/stable/2372560

[11] J. A. Reeds and L. A. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific Journal of Mathematics*, vol. 145, pp. 367–393, 1990.