Rapid Trajectory with MPC controller
AirSim NeuRIPS final solution

Jiadong Guo *Team Chuchichäschtli*
ETH Zurich
Zurich, Switzerland
jiadong.guo@outlook.com

***Abstract*—The participant [1] chose to use a combination of optimal control based planning and MPC control to solve the task 1. With known system dynamics and ground truth states, optimal control methods are well suited for generating feasible trajectories. To account for unmodeled disturbances such as drag and delays, and include input constraints, a MPC tracker is implemented to facilitate the tracking. Attached log file shows a racing record of 53.486 seconds for final tier 1.**

I. Introduction

Quadcopter is a well studied research platform for control problems. In recent years, optical control methods have been developed to allow rapid and aggressive maneuvers, while being lightweight enough to run real-time on embedded platforms. Many open-source implementations are available from previous research. With perception problem out of the way in Tier1, the participant applied minimum jerk trajectory generation from Mark Mueller et al. [1] in combination with MPC control similar to Davide Falanga et al. [2]. Both of which have open-source implementation available. The participant also intends to open-source[2] the solution post-competition.

II. System Overview

The system consist of a control and a planning module, running in separate threads. The planning module is executed whenever the controller needs a new reference trajectory, or when the state deviate too much from the planned trajectory. This could happen when drone deviates strongly from original orientation, validating several assumptions made in the simplified planner.

Control inputs are body rates and thrust values. States are chosen to be position, velocity and orientation as quaternion, same as in the MPC formulation from Davide Falanga [2]. In summary, the system can be formulated as:

$$\begin{aligned} \dot{\boldsymbol{p}}_w &= \boldsymbol{v}_w \\ \dot{\boldsymbol{v}}_w &= \boldsymbol{g} + \boldsymbol{q}_{WB} \odot \boldsymbol{T} \\ \boldsymbol{q}_{WB} &= \frac{1}{2}\Lambda(\boldsymbol{\Omega}_B)\boldsymbol{q}_{WB} \end{aligned}$$

p for position, v for velocity, q for quaternion, T for thrust, Ω for the body rates. Details and the implementation of the system refer to [2].

[1]On a side note, the participant is currently open for career opportunities.
[2]https://github.com/JD-ETH/AirSimNeuRIPS

III. Planning

Theoretically, one could also solve the global trajectory by imposing all goals as constraints, and would probably yield even better result. However, to maintain flexibility, the participant chose to use a incremental planner. Given a starting state and a goal, the planner quickly generates a feasible trajectory with nominal inputs, and forward it to the control module. Given duration of path, start and end conditions, the path can be very quickly generated as a polynoimial fitting problem. The basis of the planning module can be referred to the paper [1], here only participant's adaptations are mentioned.

A. Trajectory generation

The minimum jerk trajectory generation has a Python version available. It quickly creates trajectory between starting point and a goal, while considering the end constraints. To allow continuous fast motion in the racing setup, the goal only has constraints of zero acceleration in each dimensions(corresponding to hovering), without any limitations on end velocity.

B. Feasibility checks

On top of the input feasibility checks of body rates and thrust, the participant added an orientation constraints to disallow the drone from deviating too much from nominal thrust direction. The deviation is limited to 56 degrees. Strong tilting of the drone can violate the simplification in the planner, where motion in each translational axis is computed independently. This modification was crucial. Without this change the controller can no longer follow the planner in certain cases.

C. Binary search

To approximate optimal control given the goal location and the constraints, the planning modules runs a binary search to identify the fastest possible trajectory that conform with the given constraints, up to certain resolution. Typically, with decent boundary selection, the planning module finds the best feasible trajectory with 0.05 second tolerance after several iterations.

D. Post process

The resulting orientation of the trajectory is yaw-independent, with angular rates in inertial frame. To make things compatible with the controller, the orientation difference between current state from simulator and the initial planning state is computed, and multiplied to every states in the trajectory. This orientation is than used to convert body rates from inertial frame into actual reference control signals in body frame.

IV. Control

The MPC controller uses a linearized system model at operation point and an open-source implementation in C++ with the ACADO library [3]. The library generates C code which is highly efficient and allows run-time configurations. A python binding is added on top to call the executable through python interface. The system does not model drag forces and torques. The majority of the implementation can be used as-is, with some minor modifications.

A. Implementation details

A control frequency of 20 Hz is chosen, with a horizon of 20 steps. Every 0.05 seconds the control thread computes the next best input. Most of the weights are kept from the original formulation, with slight increase in the z position to emphasize height.

B. Input constraints

Input is thought to be bounded between 10% and twice the gravity of the drone, based on the observation that 0.5 thrust input correspond to hovering. Angular body rates of roll and pitch are limited at $\pi\ rad/s$, whereas the yaw rate is disabled. Yaw control is unnecessary for executing the optimal path. The inputs on a drone is indeed correlated: A drone can't control its body rate anymore if full thrust is engaged. The generated ACADO code can not deal with such non-linear constraints, this needs to be handled separately.

C. Using future control input

As the control input is delayed with respect to the incoming state from simulator, the control input at time step 1 instead of time step 0 is used to counteract the delay. Strictly speaking the first control input should be removed from the formulation, this however does not matter much in practice.

V. Failure handling and fine tuning

The planning thread closely observe the current state of robot, and gives the following status signals with increasing severity:

1) System is healthy, no planning needs to be done.
2) The drone has passed the target gate, new planning is required for next goal.
3) The reference trajectory in the controller is shorter than the horizon. Extend current trajectory by next immediate goal.
4) The drone has accumulated orientation error over several frames. Re-plan the path to current target. This helps correcting the discrepancy between planner and control model, which would be linearized differently otherwise.
5) Drone has saturated its control input, it's likely to become unstable. Re-plan the path to current target. This can happen when unmodeled disturbances have strong impact on the system, such as the air drag.
6) The drone has deviated from target path and is losing height. A recovery behavior is implemented trying to recover height and go back to original path.

To achieve best performance, several fine tuning was required:

1) Augment target gate poses in the z direction by a small margin. This is due to the likely height loss in highly dynamical situations.
2) To achieve some form of predictive behavior when switching from current goal to next, the planner predicts the drone's location in the immediate future and preemptively switches the planning target. This helps generating smooth path and avoid overshooting.

VI. Adaptation for Finals

Several changes had to be made for the final submission, stated below.

A. Final Tier 1

The final map is significantly more complicated and challenging, and many manual tweaks had to be made. Specifically, the following:

- Additional artificial checkpoint had to be added to avoid flying the wrong direction towards the "box" area.
- Look ahead time for switching to next target is now a parameter for each gate to be tuned. This is neede to account for vastly different velocities.

VII. Outlook

Some additional points can be implemented if this solution qualifies for the final.

- Add MPC constraints to avoid crashing into opponents or gates. This can be added as online data for generated C code from ACADO.
- Perceive the gates. To extend the current solution to Tier 3, a vision-based gate estimation needs to be implemented. The planner can than be called to quickly generate another feasible path.

References

[1] M. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," *IEEE Transactions on Robotics*, vol. 31, pp. 1–17, 10 2015.

[2] D. Falanga, P. Foehn, P. Lu, and D. Scaramuzza, "PAMPC: perception-aware model predictive control for quadrotors," *CoRR*, vol. abs/1804.04811, 2018. [Online]. Available: http://arxiv.org/abs/1804.04811

[3] B. Houska, J. Ferreau, and M. Diehl, "Acado toolkit–an open source framework for automatic control and dynamic optimization," *Optimal Control Applications and Methods*, vol. 32, pp. 298 – 312, 05 2011.