Paper Reproduction Report

Name: Jiali Sun **ID:** 3120230248

文章目录

Paper Reproduction Report
1 Paper Overview
2 Reproduction Content
3 Reproduciton Procedure
3.1 Simulation environment
3.2 Code structure explanation
3.3 Usage
3.3.1 Install Acado
3.3.2 Terminal commands
3.4 Result analysis
4 Conclusion

1 Paper Overview

The paper that is chosen to reproduce is *Perception-constrained and Motor-level Nonlinear MPC for both Underactuated and Tilted-propeller UAVs*, which is published in International Conference on Robotics and Automation(ICRA) 2020. Unlike other MPC frameworks, the MPC controller this paper proposed considers both constraints from a perceptive sensor and realistic actuator limitations that are the rotor minimum and maximum speed, which enables control in real-time the platform at a motor-torque level avoiding the use of an intermediate unconstrained trajectory tracker.

The Model Predictive Control(MPC) method of UAVs has been a popular research topic over the past decade. People compare the performance of linear and nonlinear MPC, add perception constraints to the original fundamental MPC problem. Model predictive control (MPC) is a rising control strategy used in industrial and engineering applications that involves using a mathematical model of the system being controlled to predict future behavior and determine optimal control inputs. The basic idea behind MPC is to make predictions about how the system will behave over a short time horizon, typically a few seconds to a few minutes, based on the current state of the system and the expected inputs over that time period. These predictions are then used to determine the optimal control inputs that will drive the system toward a desired outcome.

MPC problem can be formulated in a quadratic problem, then this can be solved by the solver program. This paper uses a tool called MATMPC, which is a MATLAB-based toolbox for nonlinear MPC with the external solver qpOASES. However, no source code is provided. After simulation, this paper gives experimental results, the UAV is flying along a straight line almost close to the given trajectory, while making some compromises since it has to keep the target in the camera field of view.

Normally, we take the thrusts produced by individual rotors as the input of the system, for a quadrotor, this leads to $\mathbf{u} = [f_1, f_2, f_3, f_4]$. But this paper, treats the rate of the thrusts by individual rotors as the input of the system(or the model), where $\mathbf{u} = [\dot{f}_1, \dot{f}_2, \dot{f}_3, \dot{f}_4]$. The rotor model is generally taken as $f = c_f\omega^2$, where ω is the angular velocity of the rotor and c_f is the thrust coefficient.

The framework's core formula and the core theorem of this paper can be summarized by the following equations:

The body state x_b is defined as

$$\mathbf{x_b} = [\mathbf{p}^\mathsf{T}\ \mathbf{v}^\mathsf{T}\ \eta^\mathsf{T}\ \omega^\mathsf{T}] \in \mathbb{R}^{12},$$

where p is the body position in world frame, v is the body velocity in world frame, $\eta = [\phi\ \theta\ \psi]^\mathsf{T}$ is $Z - Y - X$ euler angle representation of the body orientation, and ω is the angular velocity of the body relative to the world expressed in body frame.

The actuator state $\mathbf{x_a}$ is defined as

$$\mathbf{x_a} = \gamma \in \mathbb{R}^n,$$

where $\gamma = [f_1 \dots f_4]^\mathsf{T}$ is the forces produced by the quadrotor propellers. Consider the rate of change of the γ as the input, then

$$\dot{\gamma} = \mathbf{u}$$

Then the robot state $\mathbf{x}$ is a combination of body state x_b and actuator state $\mathbf{x_a}$, i.e.,

$$\mathbf{x} = [\mathbf{x_b}^\top \ \mathbf{x_a}^\top] \in \mathbb{R}^{12+4}$$

If the perception sensor state x_s is considered, then robot state x becomes

$$\mathbf{x} = [\mathbf{x_b}^\top \ \mathbf{x_a}^\top \ \mathbf{x_s}] \in \mathbb{R}^{12+4+1}$$

Consider the controller output

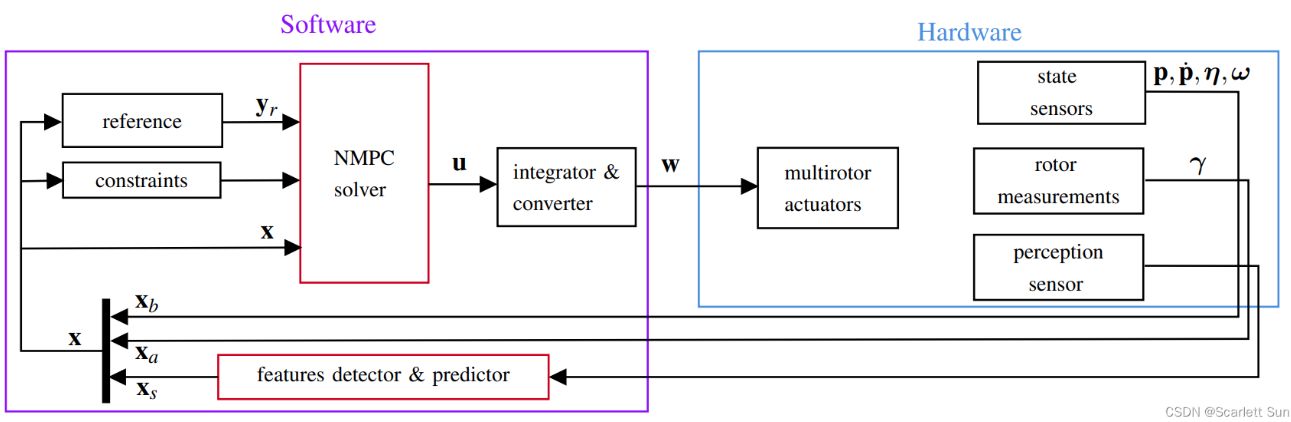
$$\mathbf{y} = [\mathbf{p}^\top \ \dot{\mathbf{p}}^\top \ \eta^\top \ \omega^\top \ \ddot{\mathbf{p}}^\top \ \dot{\omega}^\top \ \mathbf{x_s}],$$

Then the optimization problem can be formulated as:

$$\begin{aligned}
\min_{\mathbf{u}_0 \ldots \mathbf{x}_N} & \sum_{k=0}^{N} \|\mathbf{y}_k - \mathbf{y}_{r,k}\|^2_{\mathbf{Q}} \\
\text{s.t.} & \mathbf{x}_0 = \mathbf{x}(t) \\
& \mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k), \quad k = 0, 1, \ldots, N-1 \\
& \mathbf{y}_k = \mathbf{h}(\mathbf{x}_k, \mathbf{u}_k), \quad k = 0, 1, \ldots, N \\
& \underline{\gamma} \leq \gamma \leq \overline{\gamma}, \quad k = 0, 1, \ldots, N \\
& \underline{\gamma} \leq \mathbf{u}_k \leq \overline{\dot{\gamma}}, \quad k = 0, 1, \ldots, N-1 \\
& \cos\alpha \leq \mathrm{c}\beta_{i,k}, \quad k = 0, 1, \ldots, N, \quad i = 1, \ldots, P
\end{aligned}$$

The receding horizon is T and the number of shooting points is N. $\mathbf{f}$ is the dynamic function of the system. The matrix $\mathbf{Q}$ is a diagonal weight matrix. The output of the system $\mathbf{y}$ is expressed as a function $\mathbf{h}$ of the state and input. The $\mathbf{y_r}$ is provided by an external reference generator, e.g., a waypoint planner.

The block diagram of the framework is given as follows:



2 Reproduction Content

This MPC problem can be divided into three progressive stages, the first stage is the basic controller that has the same function as traditional PID controller. The second stage is replace the input of individual rotor thrusts down to motor-level thrust rates as stated as the innovation of the paper. Finally, the third is to incorporate perception constraint, but this is not novel since two years earlier than this paper, there is already paper studying the perception constraints for an MPC problem(*PAMPC: Perception-Aware Model Predictive Control for Quadrotors*,2018).

This reproduction task mainly focuses on the first stage, where the input is simplified to $\gamma = [T \ \tau_x \ \tau_y \ \tau_z]^\intercal$, the robot state $\mathbf{x} = \mathbf{x_b}$. Then the optimization problem is simplified to

$$\begin{aligned} \min_{\mathbf{u}_0...\mathbf{x}_N} & \sum_{k=0}^{N} \|\mathbf{x}_k - \mathbf{x}_{r,k}\|^2_{\mathbf{Q}} + \sum_{k=0}^{N-1} \|\mathbf{u}_k - \mathbf{u}_{r,k}\|^2_{\mathbf{R}} \\ \text{s.t.} & \mathbf{x}_0 = \mathbf{x}(t) \\ & \mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k), \quad k = 0, 1, \ldots, N-1 \\ & \underline{\gamma} \le \mathbf{u}_k \le \bar{\dot{\gamma}}, \quad k = 0, 1, \ldots, N-1 \end{aligned}$$

And different from the MATLAB implementation, the reproduction simulation will be C++ based and implemented in Robotic Operation System(ROS).

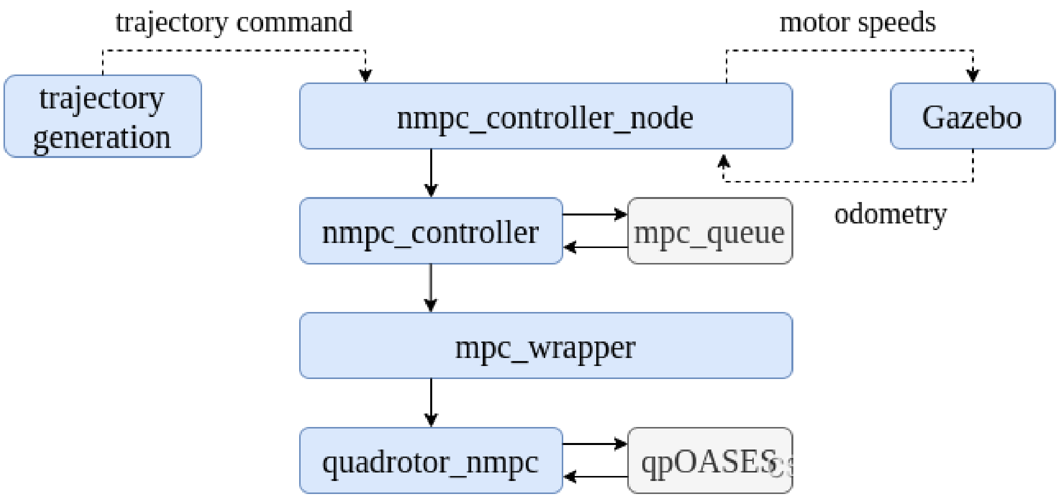
3 Reproduciton Procedure

3.1 Simulation environment

The hardware that runs the simulation is a laptop with an Intel Core i9-12900H 2.50 GHz processor and 16GB 4800 MHz DDR5. The software environment is Ubuntu 20.04 with Robot Operation System(ROS). The controller will be tested and verified in the Gazebo simulation environment based on the utilities and plugins provided by RotorS, a popular MAV simulator. The solver is chosen as ACADO, which is an open-source software framework originally developed at the Swiss Federal Institute of Technology in Zurich (ETH Zurich). The software includes a suite of algorithms and tools for formulating optimization problems, generating efficient numerical code, and solving the resulting optimization problems using state-of-the-art numerical optimization techniques. The code will be in C++ and in the form of ROS packages.

3.2 Code structure explanation

The code can be categorized in the following modules.



First, in the `model` folder, define the MPC model in a C++ source file, e.g., `quadrotor_nmpc.cpp`, and then follow the instructions in `README.m` file to generate the C++ code for the computation of the MPC problem. The external solver `qpOASES` should also be included as in the `externals` folder. Then, the `mpc_wrapper.cpp` is used to encapsulate the interface code generated by ACADO. In `nmpc_controller.cpp`, the current state and the reference state is set for the controller to run, and results of each control loop is updated and stored here. The `nmpc_controller_node` is mainly responsible for communication purpose, it will interact with the trajectory generation module and the Gazebo simulator. It will set the controller parameters like the weights and the boundaries from ROS parameter server. It subscribes to odometry for estimated value and trajectory command. It publishes the controller output as the motor speeds to Gazebo simulator. It runs the controller at a given rate using a timer.

The parameters are set as follows:

```
# weights for states
q_position: {x: 20, y: 20, z: 50}
q_velocity: {x: 10, y: 10, z: 10}
q_attitude: {x: 50, y: 50, z: 50}
q_angular_rate: {x: 10, y: 10, z: 10}

# weights for inputs
r_thrust: 0.1
r_tau: {x: 1, y: 1, z: 1}

# limits
max_thrust: 18 # [N]
min_thrust: 0.1 # [N]
max_tau: {x: 0.05, y: 0.05, z: 0.05} # [Nm]

# time
```

```
controller_frequency: 100 # [Hz]
queue_dt: 0.01 # [s]
prediction_sampling_time: 1 # [s]
number_of_samples: 10 # N in the equation
```

3.3 Usage

3.3.1 Install Acado

See below references from my blog:
Configure Acado on Linux
Generate MPC C++ code using Acado

3.3.2 Terminal commands

Create a workspace

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src
catkin_init_workspace  # initialize your catkin workspace
wstool init
```

Download the rotors_simulator packages

```
cd ~/catkin_ws/src
git clone git@github.com:ethz-asl/rotors_simulator.git
git clone git@github.com:ethz-asl/mav_comm.git
```

Unzip and copy the code packages to `~/catkin_ws/src`, and then build the workspace

```
cd ~/catkin_ws
catkin build
```

Open one terminal, and run commands below to start Gazebo and the controller

```
source ./devel/setup.bash
roslaunch nmpc_controller nonlinear_mpc_sim.launch
```

Open another terminal, and run commands below to publish the trajectory

```
source ./devel/setup.bash
rosrun nmpc_controller waypoint_publisher_5rd_polynomial \
__ns:=/ardrone 0 0 0 0 1 0.08 0 0 10 # to hover
```

> **Note:**
>
> 1. arguments: <x_d> <x_sen> <y_d> <y_sen> <z_d> <z_sen> <yaw_d> <yaw_sen>
> 2. __ns:=/ardrone is to set the namespace of the node, do not change this.

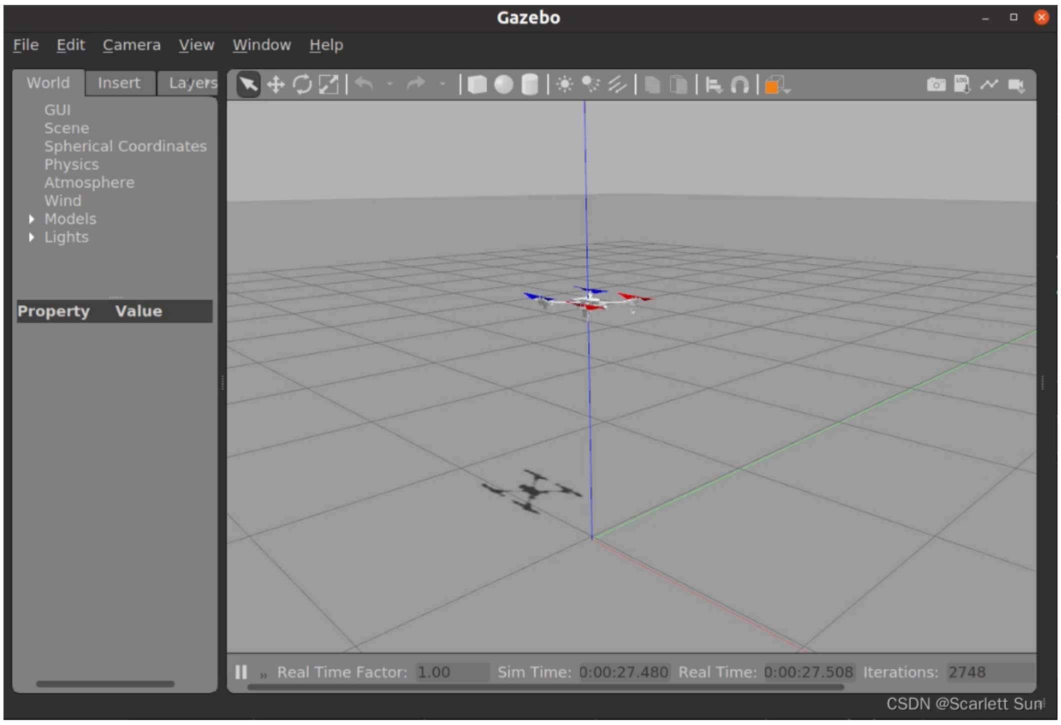
And then the UAV will fly according to the commanded height of 1m as shown in the attached video. After the UAV reaches a stable state, more maneuvers can be performed by changing the arguments.

Open another terminal to inspect the demanded trajectory, here a data visualiztion panel tool called `foxglove` is used. For more information, refer to its website

```
source ./devel/setup.bash
rosrun foxglove_bridge foxglove_bridge
```

3.4 Result analysis

To test the function of the simulation code, a series of movement is performed in gazebo. The trajectory is generated as fifth order polynomials.



As can be seen in the video, for each motion, the actual trajectory converges to the reference, which indicates that the formulated MPC problem is solvable with given weights, limits, time horizon and number of samples.

4 Conclusion

The report summarizes the reproduction procedure of paper *Perception-constrained and Motor-level Nonlinear MPC for both Underactuated and Tilted-propeller UAVs*. This paper proposed a motor-level NMPC framework that considers perception constraints. Its implementation can be divided into three stages. This report mainly focuses on the first stage, where the torque and thrust are treated as the input, and their limits will be the boundary conditions of the optimization problem.This reproduction utilizes ACADO as the solver, and the code is written in C++ with simulation environment Gazebo in ROS. This report formulates the optimization problem and gives introduction to the code structure. The usage of the code is also explained, and the analysis of the simulation result of a hovering example is given.