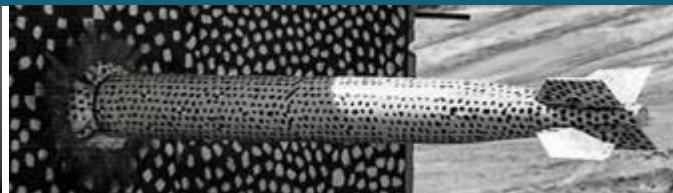




Sandia  
National  
Laboratories

# CUDA for Rapid Controller Robustness Evaluation: A Tutorial



Manan Gandhi, Rachel Schlossman, Kyle A. Williams,  
Ryan Melzer, Julie Parish

2021 Conference on Decision and Control  
December 16, 2021

Copyright © 2021 National Technology & Engineering Solutions of Sandia, LLC (NTESS). This presentation has been authored by NTESS under Contract No. DE-NA0003525 with the U.S. Department of Energy/National Nuclear Security Administration. The United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this presentation, or allow others to do so, for United States Government purposes. Published by the American Institute of Aeronautics and Astronautics, Inc., with permission.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

For audio, see  
[https://youtu.be/aj7ti\\_QfPmk](https://youtu.be/aj7ti_QfPmk)



Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.



GATE (GPU-Accelerated Trajectory Evaluation) is a tool designed for rapid Monte-Carlo simulations for complex dynamical systems.

The primary benefit of GATE is a *fast, plug-and-play* approach to underlying dynamics, controllers, and perturbations.

This tool allows for fast evaluation of controller robustness against a wide variety of perturbations and disturbances through parallel simulations (i.e., rollouts).

Our case study for a quadrotor system shows that GATE outperforms other parallel simulation technologies using CUDA C/C++.



# CUDA Overview





CUDA (Compute Unified Device Architecture) is a programming interface based on C which allows users to take advantage of massive parallelism offered by GPU's for general-purpose programming.

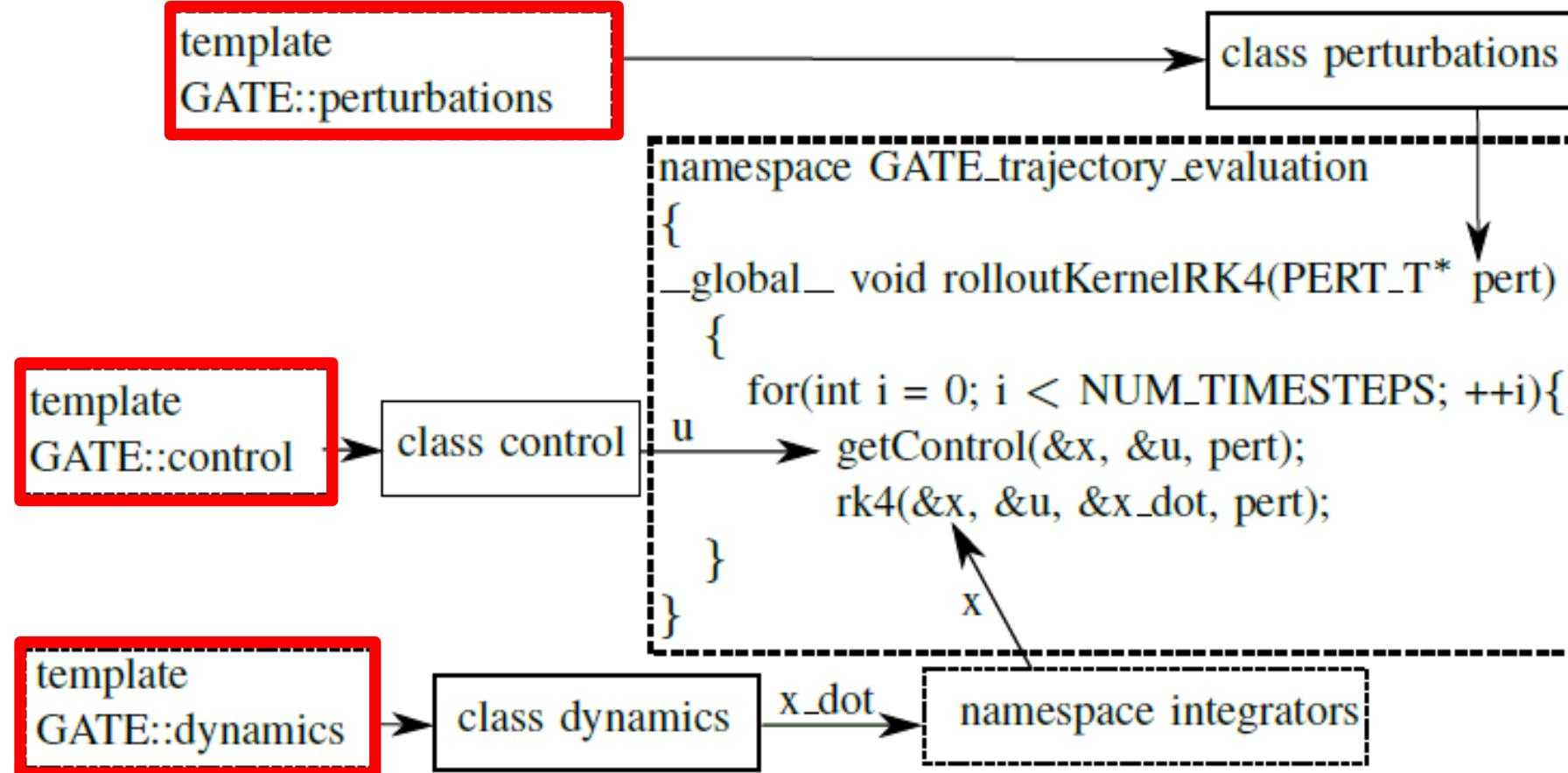
The key points to be aware of is how memory is handled between the CPU *host* and the GPU *device*, and how many individual processes are run via the CUDA *kernel*.

GATE simplifies the process of handling memory and has a custom kernel written to execute multiple trajectories at once, along with complex control systems.



# GATE Architecture

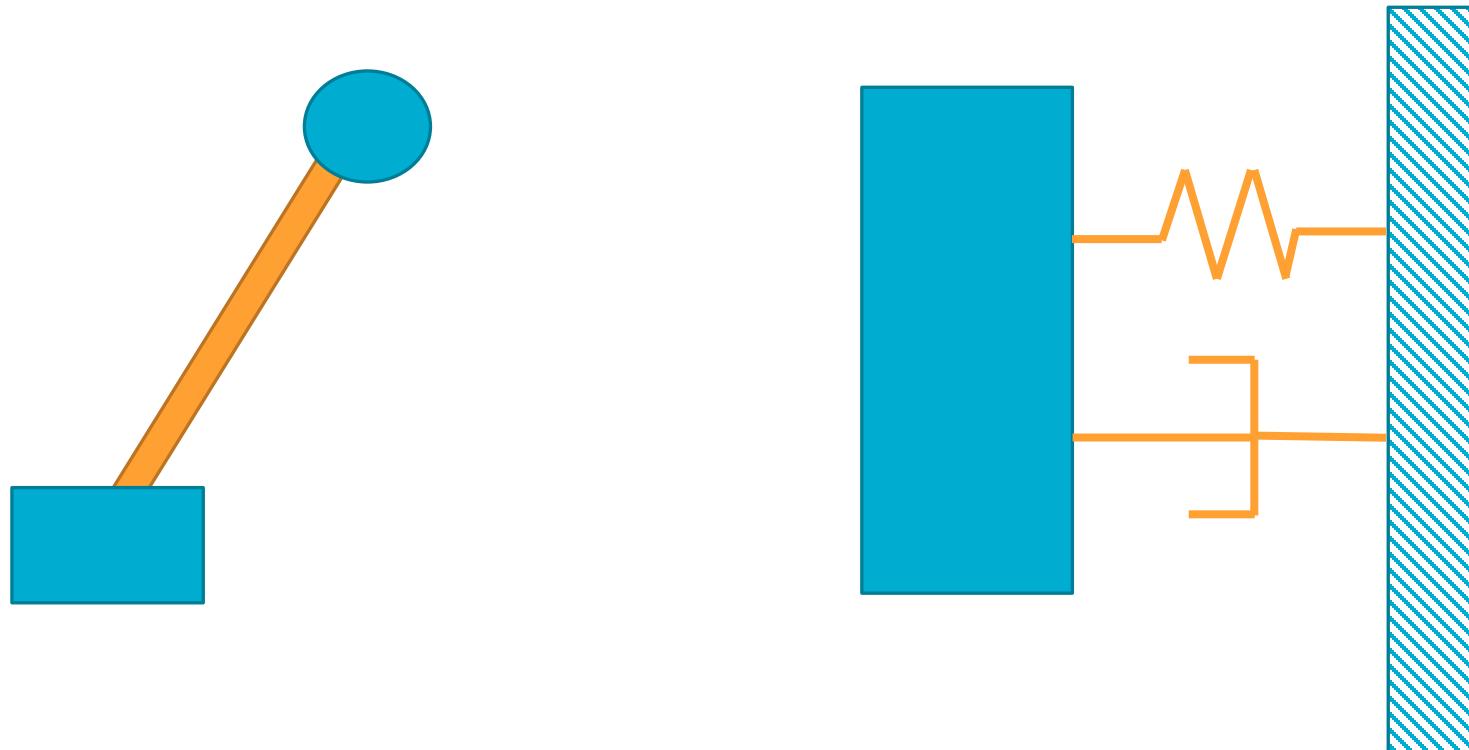






## ■ Dynamics:

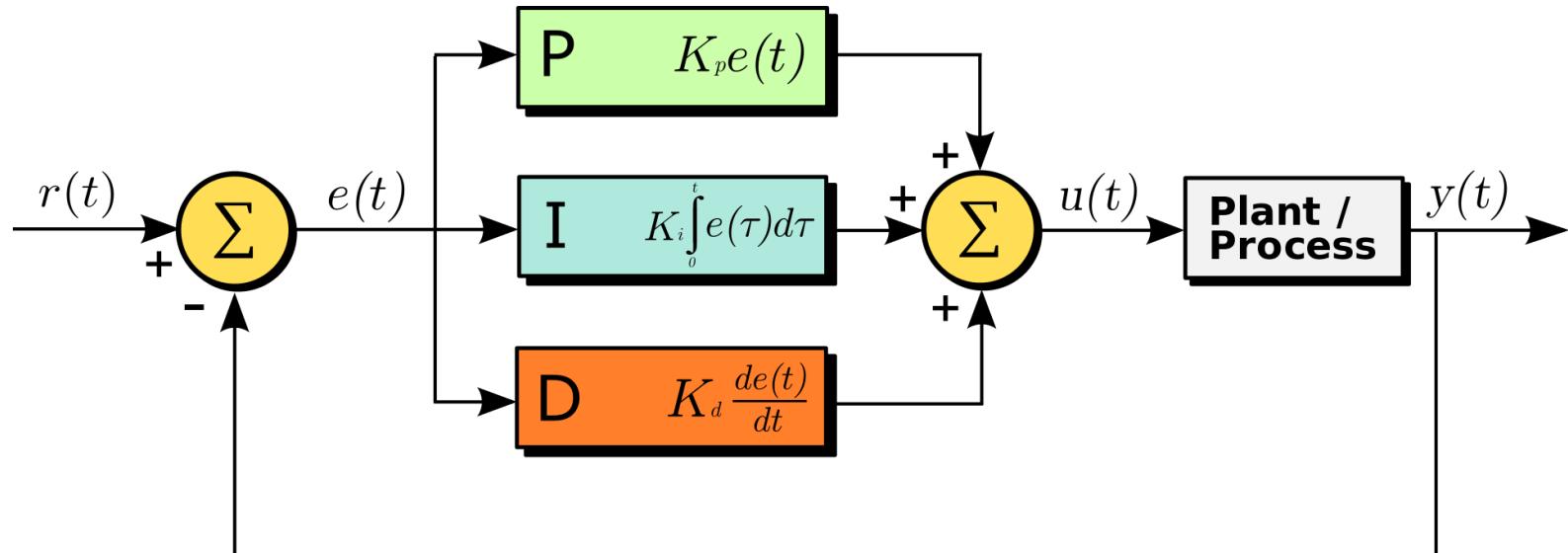
- Implements state derivative computation given a control input and set of perturbations.
- Can be modified to represent multi-phase, hybrid systems.





## ■ Controllers:

- Implements various control strategies designed by the end user.
- Support for per-rollout parameters allows for adaptation of the controller based on the simulation.

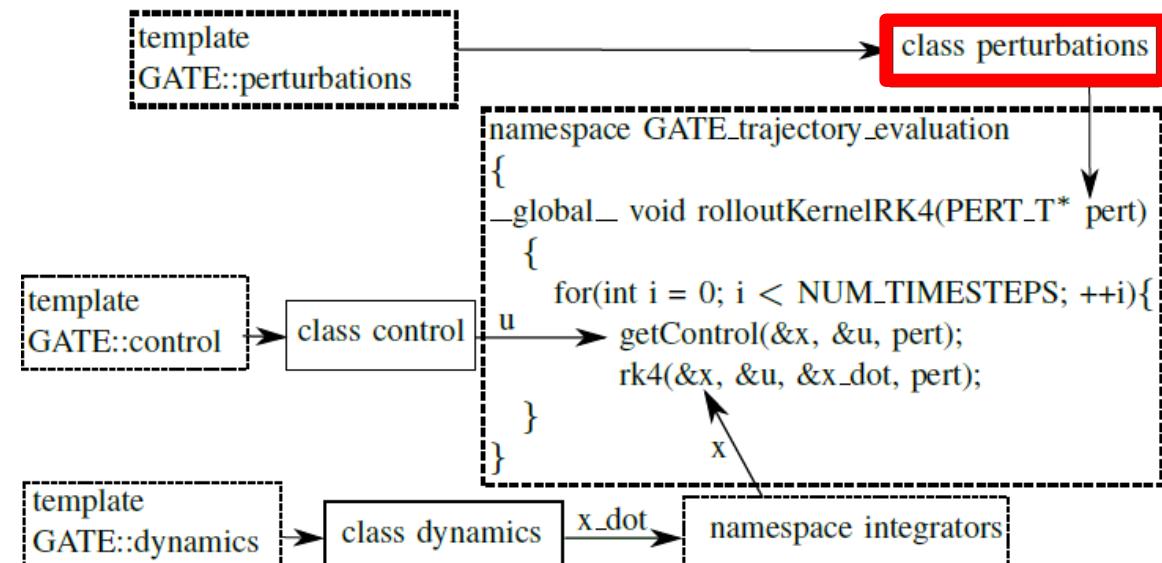
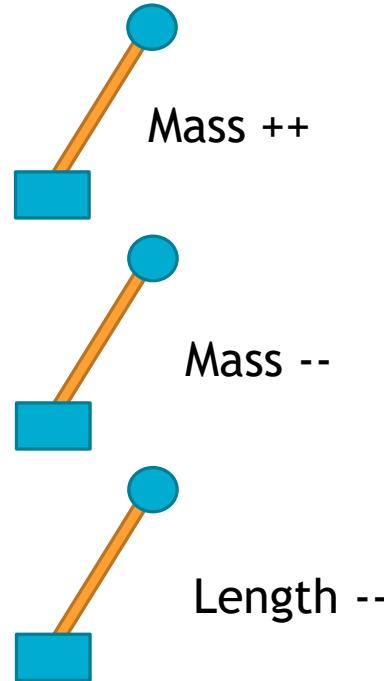


Source: Wikipedia Commons: [https://upload.wikimedia.org/wikipedia/commons/thumb/4/43/PID\\_en.svg/1920px-PID\\_en.svg.png](https://upload.wikimedia.org/wikipedia/commons/thumb/4/43/PID_en.svg/1920px-PID_en.svg.png)



## ■ Perturbation

- Implements perturbations in the initial condition and in the control channel.
- Support for variation in parameters in both time horizon and number of rollouts.





# Implementation Details



# Getting Started

[Code](#)[Issues](#)[Pull requests](#)[Actions](#)[Projects](#)[Wiki](#)[Security](#)[Insights](#)[Settings](#)[master](#)[1 branch](#)[0 tags](#)[Go to file](#)[Add file](#)[Code](#)[rschlos-sn1 Merge pull request #1 from mgandhi635/master](#)[...](#)

59cdf39 9 days ago

5 commits

[eigen-3.3.8](#)

Add Eigen Core back.

18 days ago

[gate](#)

pushing public code

last month

[googletest-master](#)

pushing public code

last month

[test](#)

pushing public code

last month

[.gitignore](#)

pushing public code

last month

[CMakeLists.txt](#)

pushing public code

last month

[CMakeSettings.json](#)

pushing public code

last month

[LICENSE](#)

license and readme

2 months ago

[README.md](#)

readme update

last month

[README.md](#)

## GATE-PUBLIC

<https://github.com/sandialabs/gate-public>

### About



No description, website, or topics provided.

[Readme](#)[MIT License](#)

### Releases

No releases published

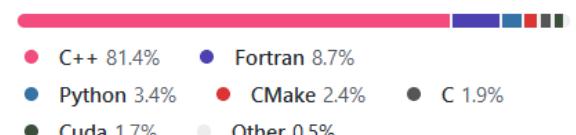
[Create a new release](#)

### Packages

No packages published

[Publish your first package](#)

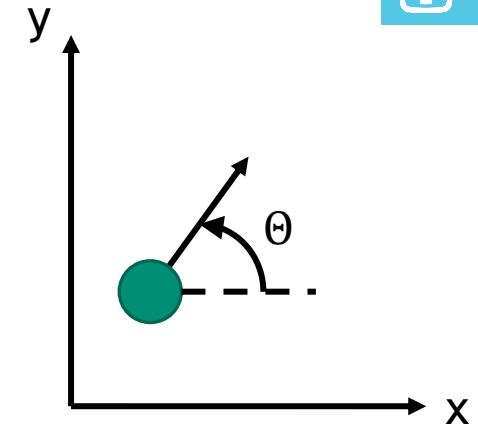
### Languages



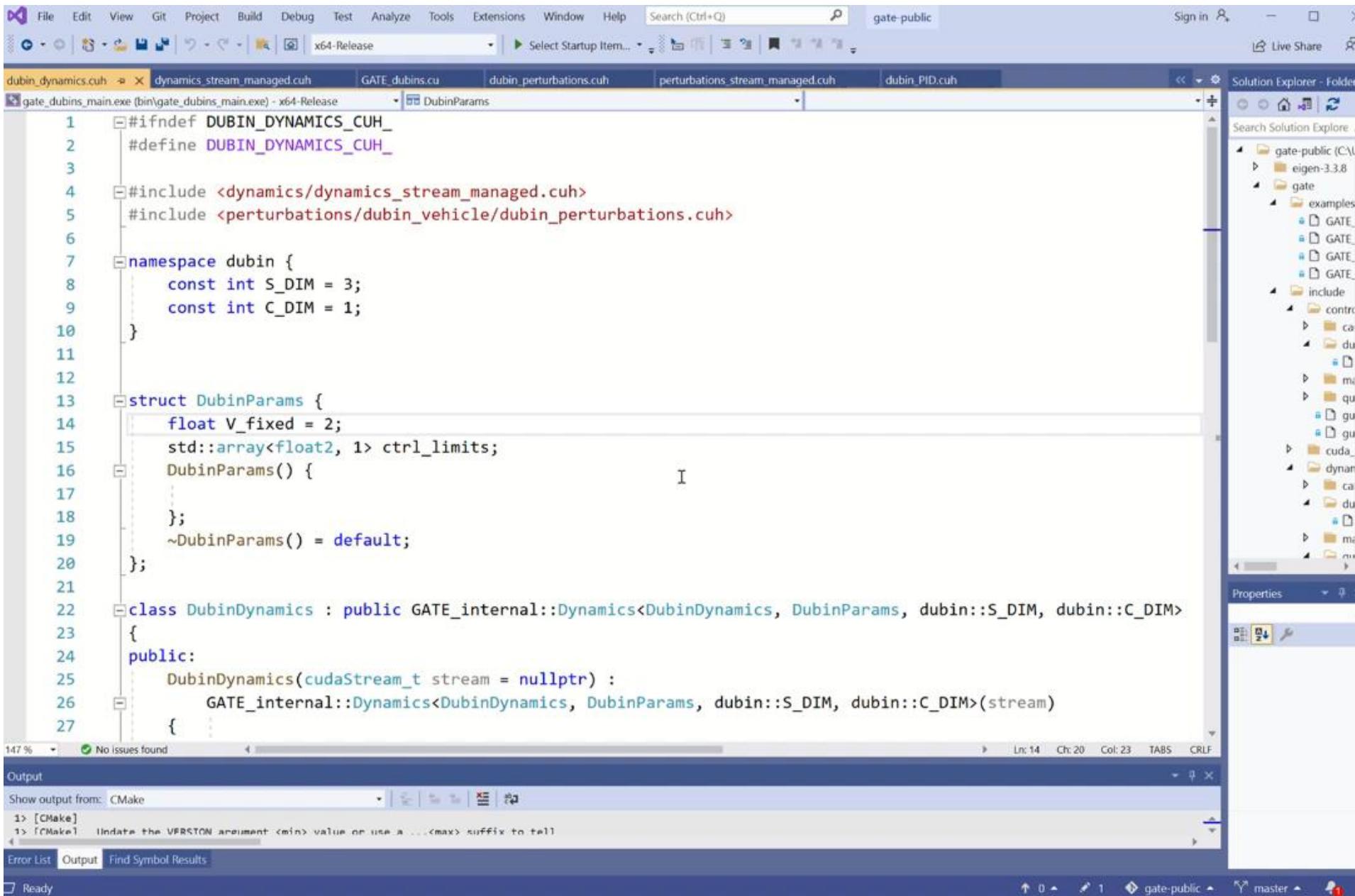
# Example: Dubins Vehicle



Dynamics	Perturbations	Control
$\dot{X}(t) = V \cos(\theta(t))$ $\dot{Y}(t) = V \sin(\theta(t))$ $\dot{\theta}(t) = \omega_{cmd}(t) + w(t)$ <p>where:</p> $\bar{x} = [X \quad Y \quad \theta]^T$	<p>Control input:</p> $w \sim \mathcal{N}(0, \sigma_w^2)$ <p>Initial conditions:</p> $\bar{x}_0 \sim \mathcal{N}(\mu_{\bar{x}}, \sigma_{\bar{x}}^2)$	$\omega_{cmd} = K_p \Delta \theta$ <p>where:</p> $\Delta \theta \triangleq (\theta_{des} - \theta)$ $\theta_{des} = \text{atan2}(\Delta y, \Delta x)$ $\Delta x \triangleq (X_{des} - X)$ $\Delta y \triangleq (Y_{des} - Y)$



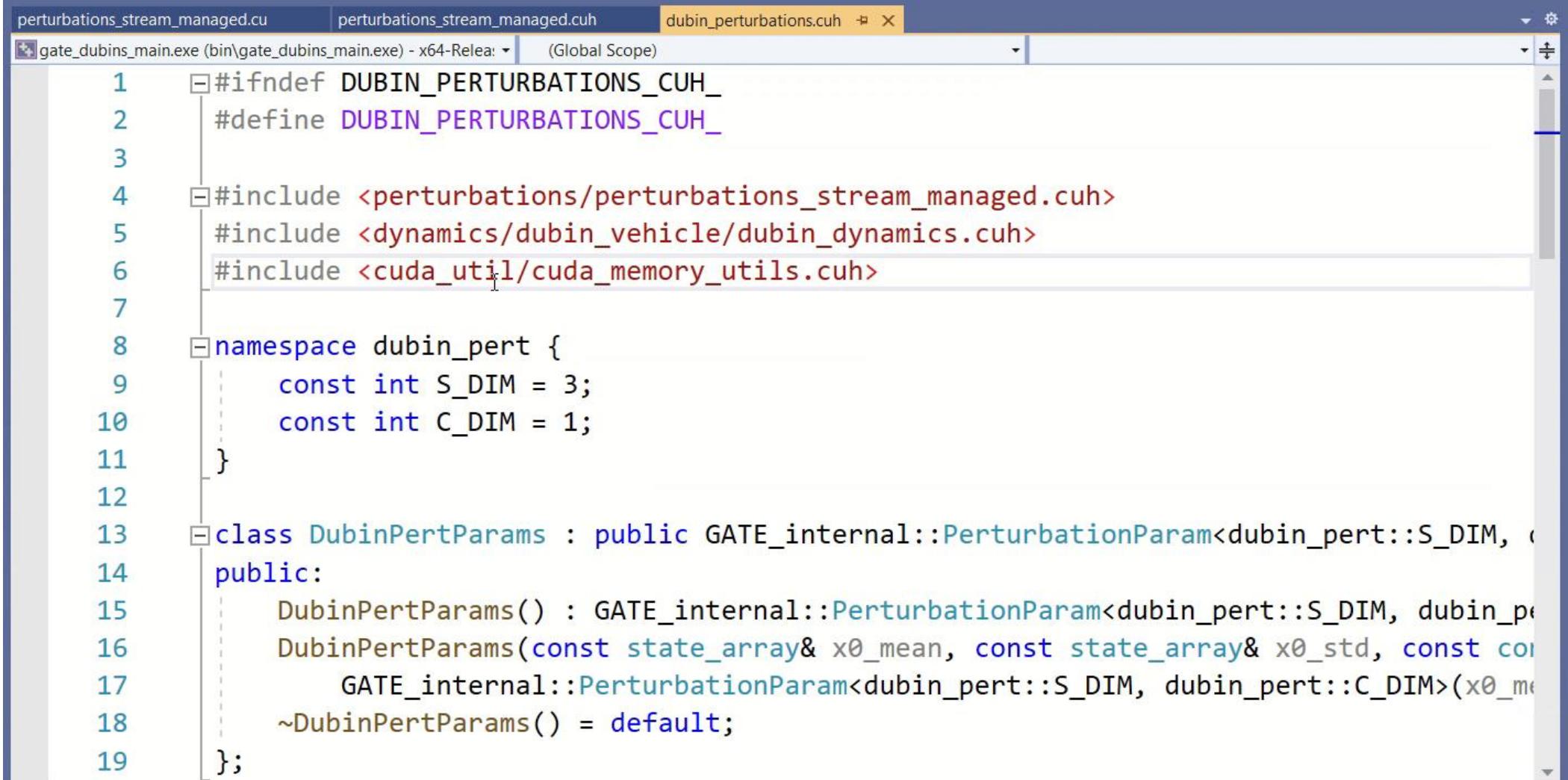
# Dubins Dynamics Class



The screenshot shows the Microsoft Visual Studio IDE interface with the following details:

- File Menu:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Search Bar:** Search (Ctrl+Q) with a magnifying glass icon.
- Project Explorer:** Shows the solution structure: `gate-public` (C:\), `eigen-3.3.8`, `gate`, `examples` (containing `GATE_` variants), `include` (containing `control` and `cuda_` subfolders), and `dynamics` (containing `ca`, `du`, `mi`, `qu`, `gu`, and `gu2` subfolders).
- Solution Explorer:** Shows the current project structure: `gate-public` (bin\gate\_dubins\_main.exe - x64-Release), `dubin_dynamics.cuh`, `dynamics_stream_managed.cuh`, `GATE_dubins.cu`, `dubin_perturbations.cuh`, `perturbations_stream_managed.cuh`, and `dubin_PID.cuh`.
- Properties Window:** Visible on the right side of the interface.
- Code Editor:** Displays the `dubin_dynamics.cuh` file content, which includes the `DUBIN_DYNAMICS_CUH_` macro, includes for `dynamics_stream_managed.cuh` and `dubin_perturbations.cuh`, a `namespace dubin` block with constants `S_DIM` and `C_DIM`, a `DubinParams` struct with a constructor and destructor, and a `DubinDynamics` class that inherits from `GATE_internal::Dynamics`.
- Output Window:** Shows CMake build logs, including the message: `1> [CMake] 1> [CMake] Update the VERSTON argument <min> value or use a ...<max> suffix to tell`.
- Bottom Bar:** Shows tabs for Error List, Output, and Find Symbol Results, and a status bar indicating the file is ready.

# Dubins Perturbations Class



```
perturbations_stream_managed.cu | perturbations_stream_managed.cuh | dubin_perturbations.cuh
gate_dubins_main.exe (bin\gate_dubins_main.exe) - x64-Release | (Global Scope)

1 #ifndef DUBIN_PERTURBATIONS_CUH_
2 #define DUBIN_PERTURBATIONS_CUH_
3
4 #include <perturbations/perturbations_stream_managed.cuh>
5 #include <dynamics/dubin_vehicle/dubin_dynamics.cuh>
6 #include <cuda_util/cuda_memory_utils.cuh>
7
8 namespace dubin_pert {
9     const int S_DIM = 3;
10    const int C_DIM = 1;
11 }
12
13 class DubinPertParams : public GATE_internal::PerturbationParam<dubin_pert::S_DIM, dubin_pert::C_DIM> {
14 public:
15     DubinPertParams() : GATE_internal::PerturbationParam<dubin_pert::S_DIM, dubin_pert::C_DIM>()
16     DubinPertParams(const state_array& x0_mean, const state_array& x0_std, const covariance_array& cov)
17         GATE_internal::PerturbationParam<dubin_pert::S_DIM, dubin_pert::C_DIM>(x0_mean, x0_std, cov);
18     ~DubinPertParams() = default;
19 }
```

161 %

No issues found

Ln: 6 Ch: 20 TABS CRLF

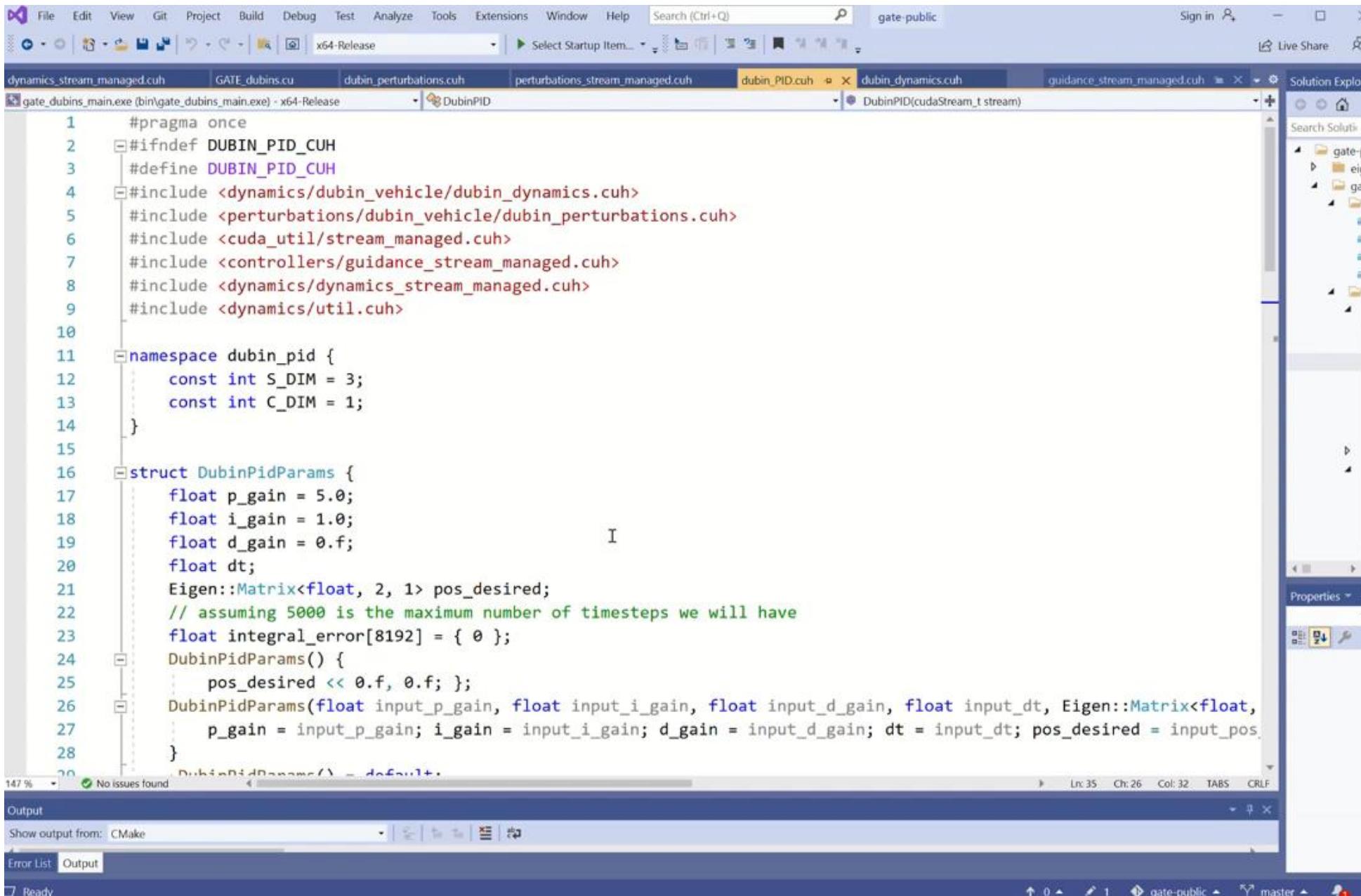
Output

Show output from: Source Control - Git



Error List Output

# Dubins Control Class

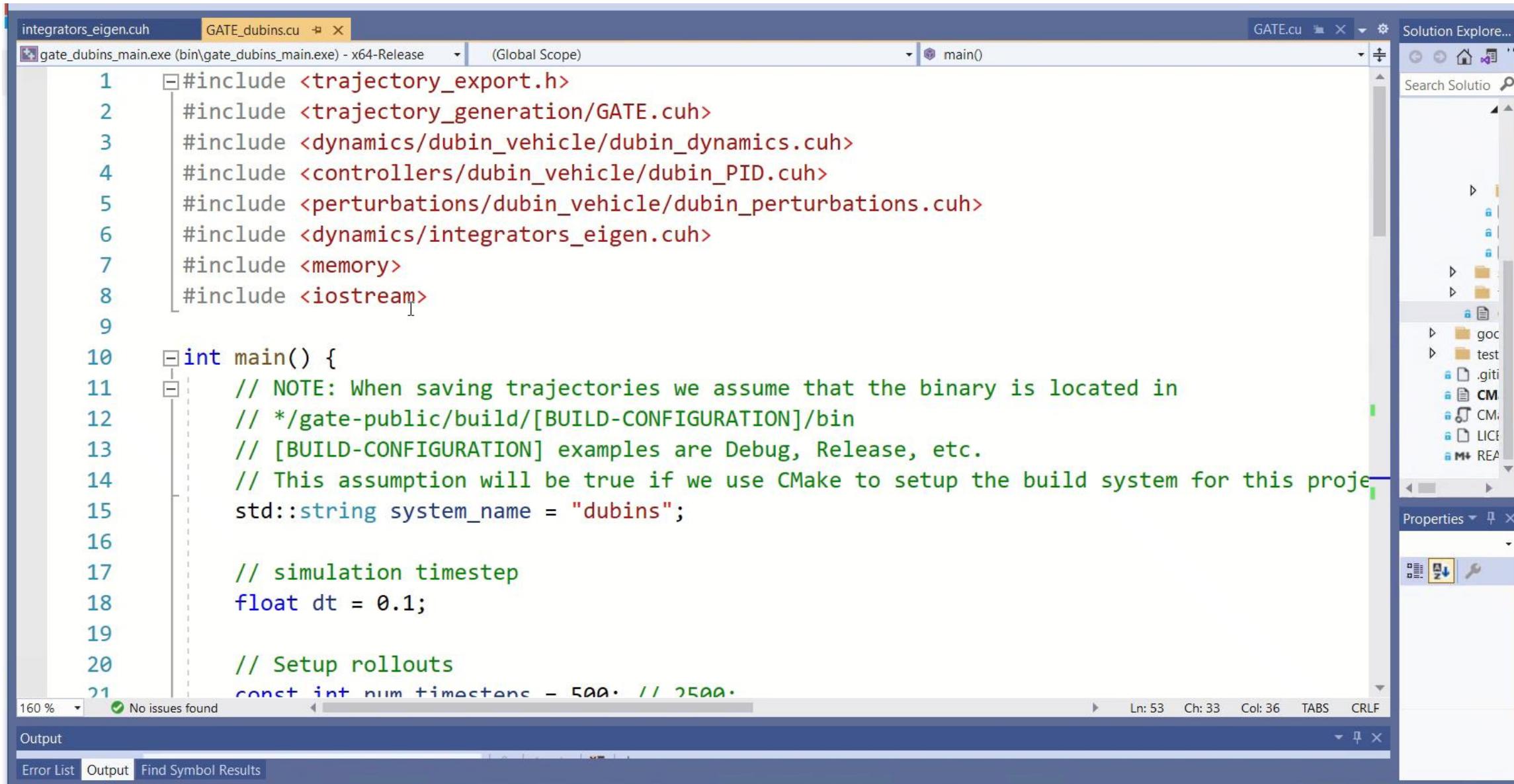


The screenshot shows the Microsoft Visual Studio IDE interface with the following details:

- File Bar:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Search Bar:** Search (Ctrl+Q).
- Active Tab:** dubin\_PID.cuh.
- Solution Explorer:** Shows the project structure with files like gate\_dubins\_main.exe, GATE\_dubins.cu, dubin\_perturbations.cuh, perturbations\_stream\_managed.cuh, dubin\_PID.cuh, dubin\_dynamics.cuh, and guidance\_stream\_managed.cuh.
- Code Editor:** Displays the code for the Dubin PID class, including the struct definition and constructor implementation.
- Properties Window:** Shows the properties for the selected file.
- Output Window:** Shows the build output.
- Error List:** Shows the list of errors.
- Ready:** Status bar indicator.

```
1 #pragma once
2 #ifndef DUBIN_PID_CUH
3 #define DUBIN_PID_CUH
4 #include <dynamics/dubin_vehicle/dubin_dynamics.cuh>
5 #include <perturbations/dubin_vehicle/dubin_perturbations.cuh>
6 #include <cuda_util/stream_managed.cuh>
7 #include <controllers/guidance_stream_managed.cuh>
8 #include <dynamics/dynamics_stream_managed.cuh>
9 #include <dynamics/util.cuh>
10
11 namespace dubin_pid {
12     const int S_DIM = 3;
13     const int C_DIM = 1;
14 }
15
16 struct DubinPidParams {
17     float p_gain = 5.0;
18     float i_gain = 1.0;
19     float d_gain = 0.f;
20     float dt;
21     Eigen::Matrix<float, 2, 1> pos_desired;
22     // assuming 5000 is the maximum number of timesteps we will have
23     float integral_error[8192] = { 0 };
24     DubinPidParams() {
25         pos_desired << 0.f, 0.f;
26     }
27     DubinPidParams(float input_p_gain, float input_i_gain, float input_d_gain, float input_dt, Eigen::Matrix<float,
28         p_gain = input_p_gain; i_gain = input_i_gain; d_gain = input_d_gain; dt = input_dt; pos_desired = input_pos;
29     }
```

# Running the Simulation



```
integrators_eigen.cuh GATE_dubins.cu GATE.cu Solution Explore...
gate_dubins_main.exe (bin\gate_dubins_main.exe) - x64-Release (Global Scope) main()
1 #include <trajectory_export.h>
2 #include <trajectory_generation/GATE.cuh>
3 #include <dynamics/dubin_vehicle/dubin_dynamics.cuh>
4 #include <controllers/dubin_vehicle/dubin_PID.cuh>
5 #include <perturbations/dubin_vehicle/dubin_perturbations.cuh>
6 #include <dynamics/integrators_eigen.cuh>
7 #include <memory>
8 #include <iostream>
9
10 int main() {
11     // NOTE: When saving trajectories we assume that the binary is located in
12     // */gate-public/build/[BUILD-CONFIGURATION]/bin
13     // [BUILD-CONFIGURATION] examples are Debug, Release, etc.
14     // This assumption will be true if we use CMake to setup the build system for this project
15     std::string system_name = "dubins";
16
17     // simulation timestep
18     float dt = 0.1;
19
20     // Setup rollouts
21     const int num_timesteps = 500. // 2500.
```

160% No issues found Ln: 53 Ch: 33 Col: 36 TABS CRLF

Output

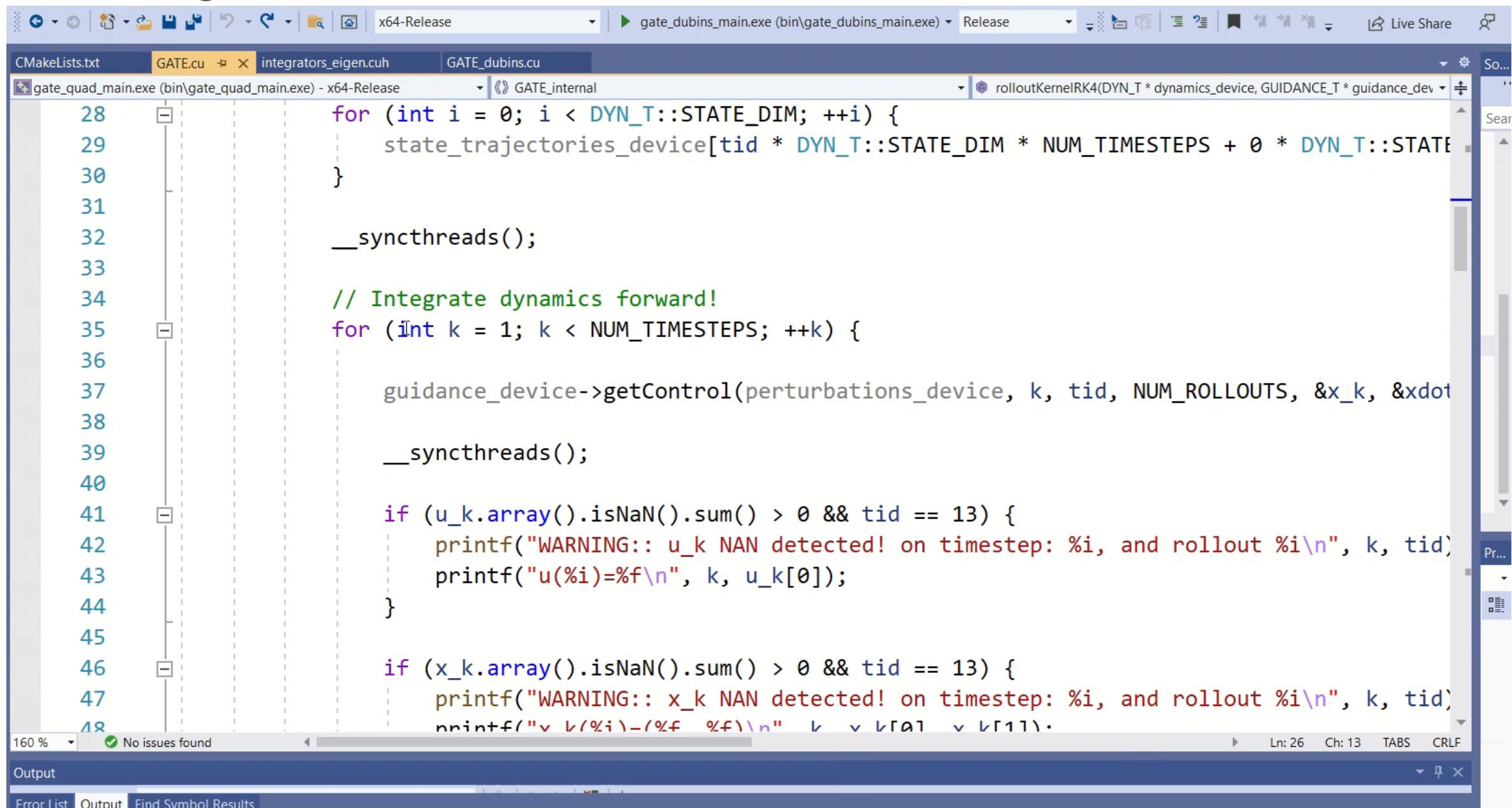
Error List Output Find Symbol Results

Search Solution

Properties

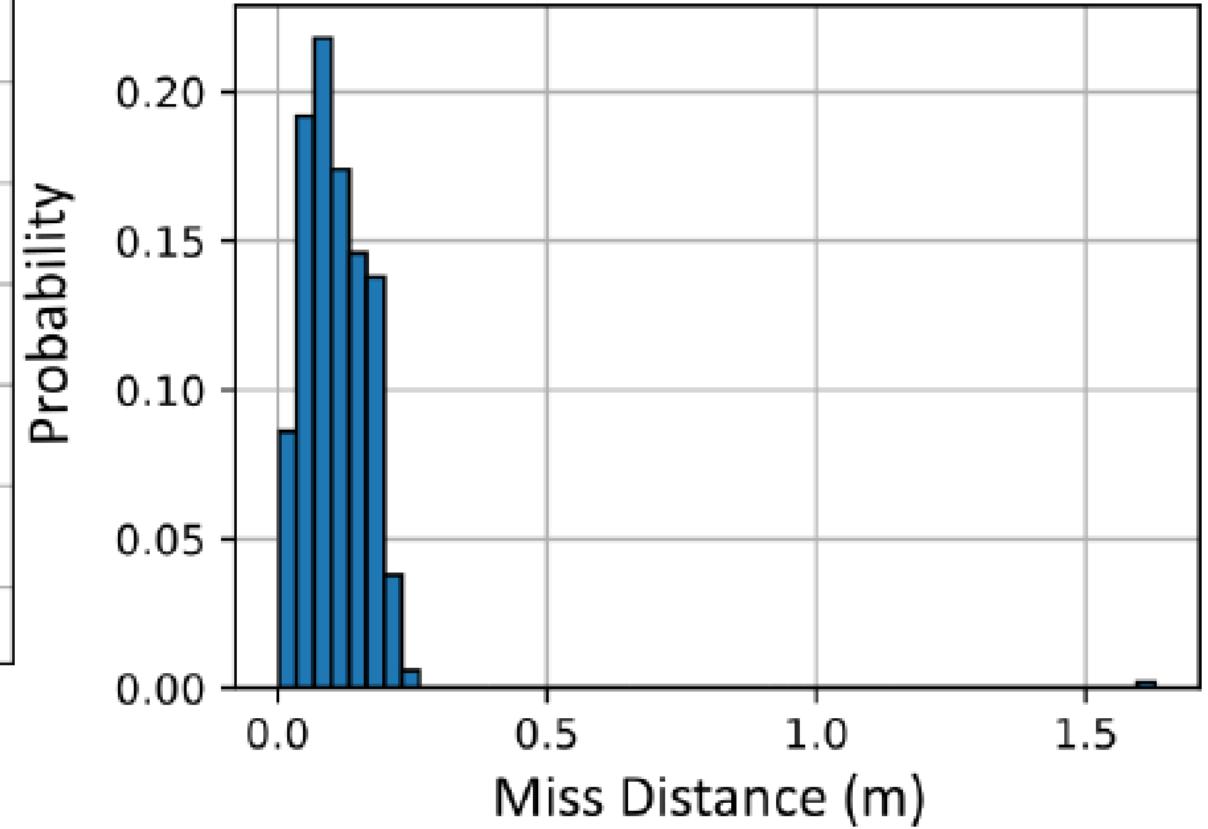
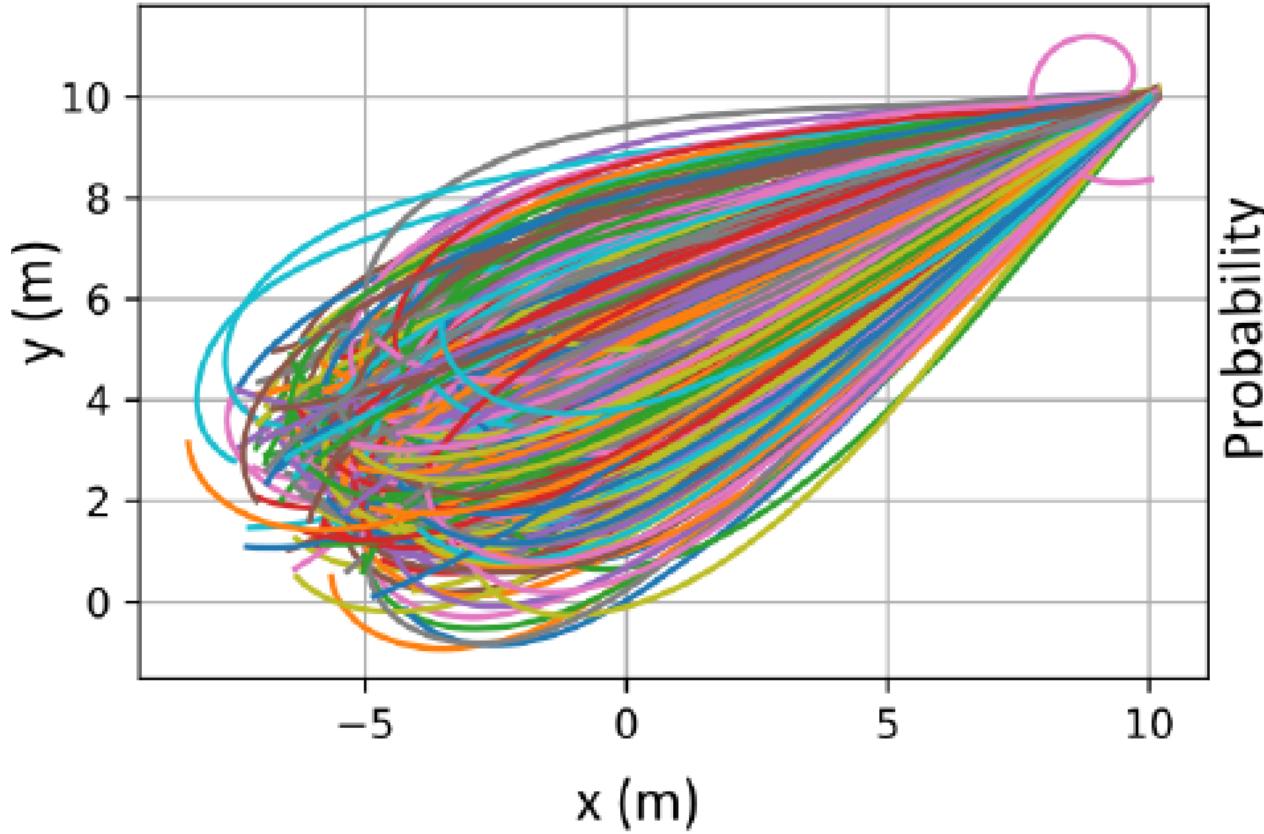
goc test .giti CM CM LICENSE README

# Running the Simulation



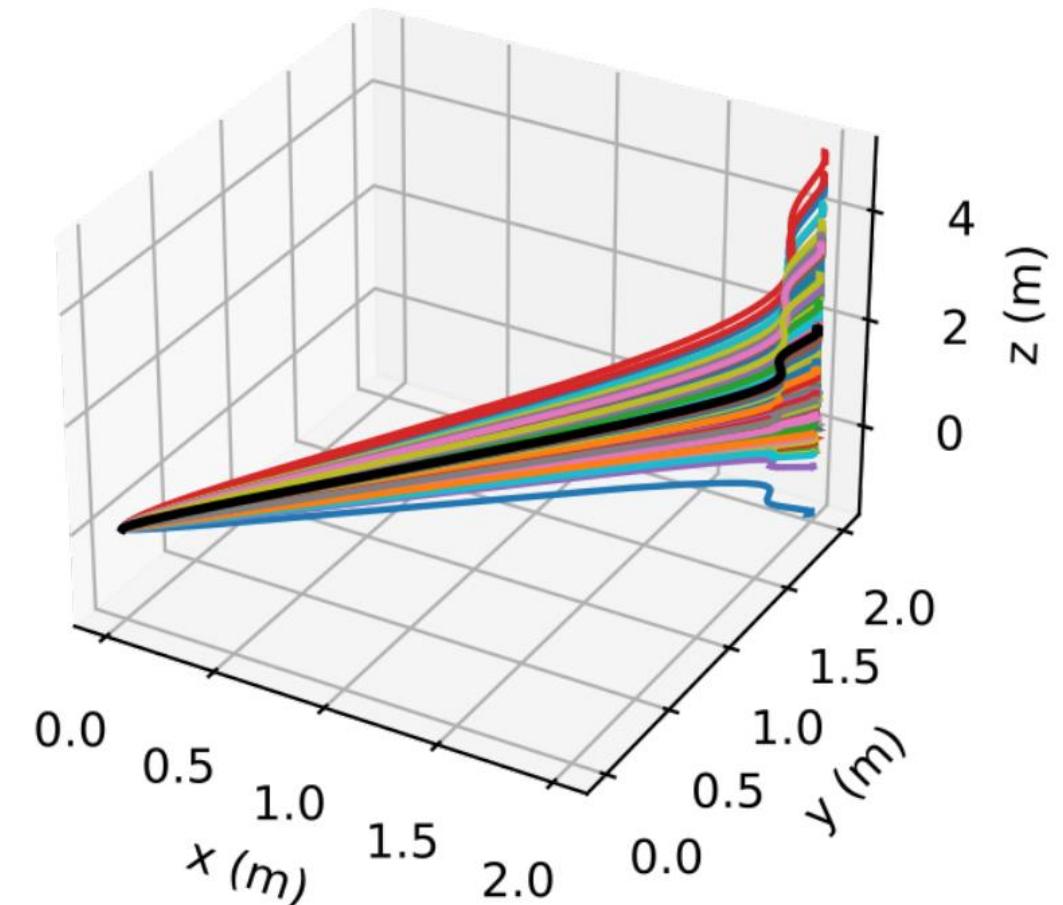
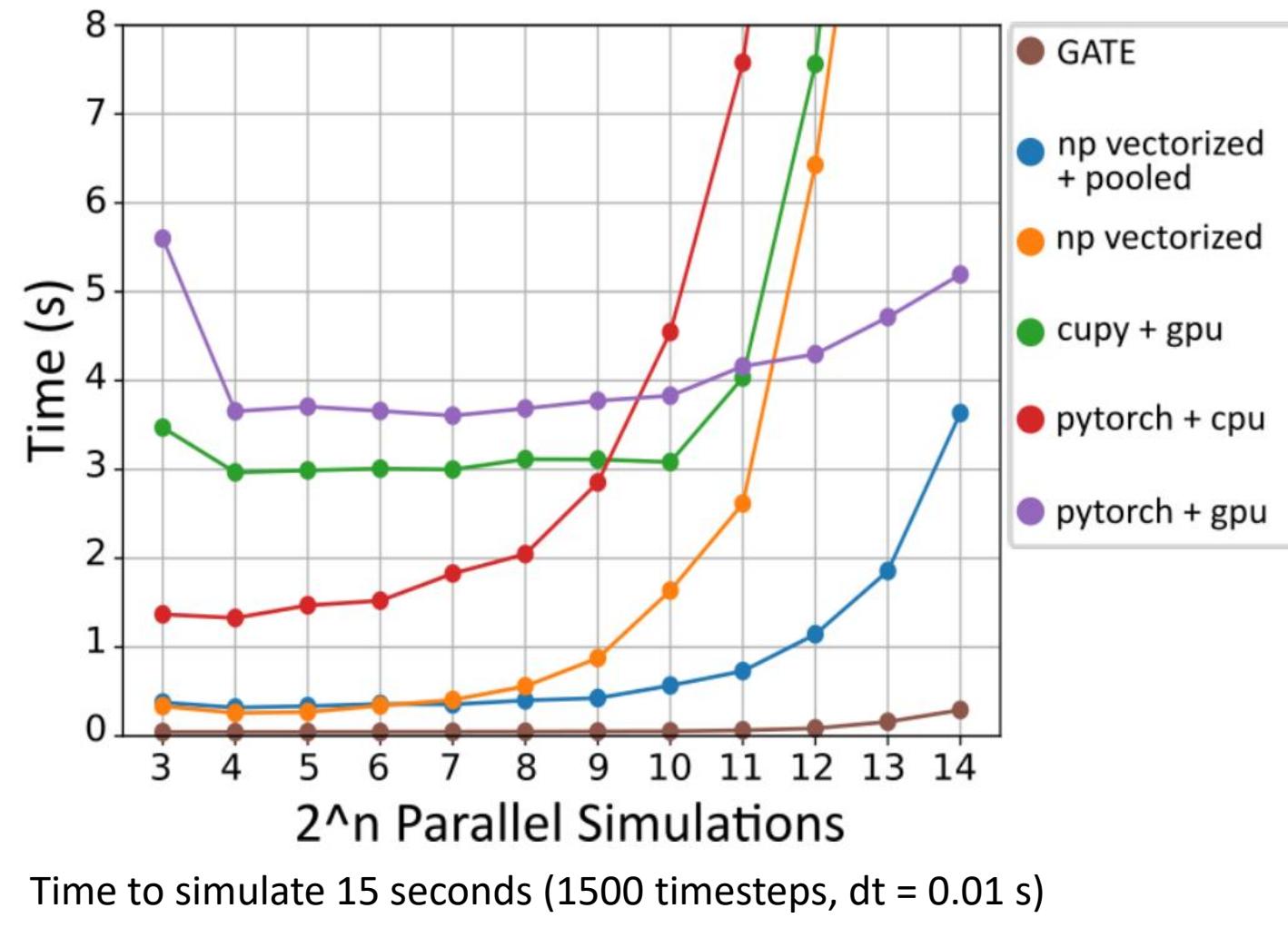
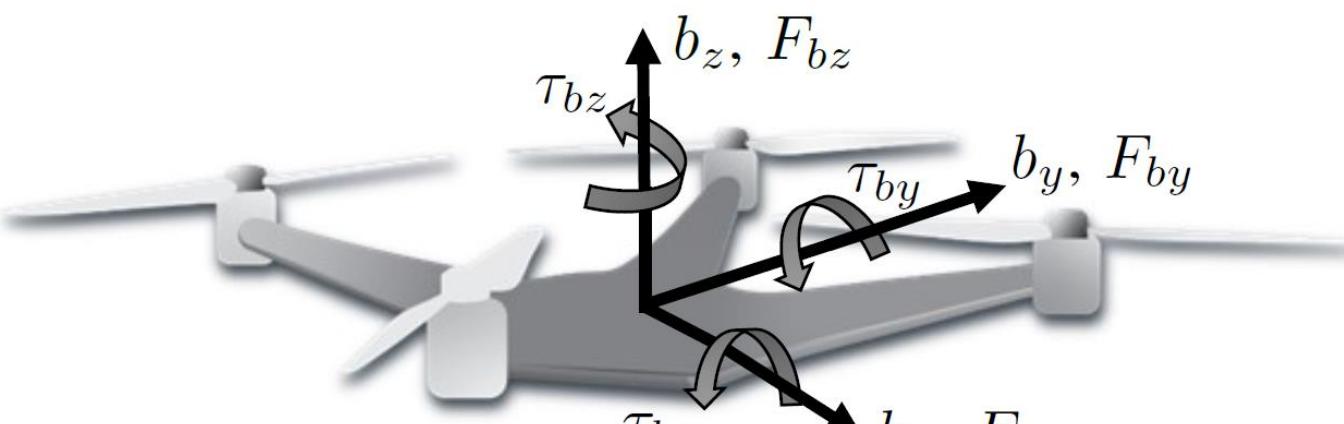
```
28     for (int i = 0; i < DYN_T::STATE_DIM; ++i) {
29         state_trajectories_device[tid * DYN_T::STATE_DIM * NUM_TIMESTEPS + 0 * DYN_T::STATE_DIM + i] = state;
30     }
31
32     __syncthreads();
33
34     // Integrate dynamics forward!
35     for (int k = 1; k < NUM_TIMESTEPS; ++k) {
36
37         guidance_device->getControl(perturbations_device, k, tid, NUM_ROLLOUTS, &x_k, &xdot);
38
39         __syncthreads();
40
41         if (u_k.array().isNaN().sum() > 0 && tid == 13) {
42             printf("WARNING:: u_k NAN detected! on timestep: %i, and rollout %i\n", k, tid);
43             printf("u(%i)=%f\n", k, u_k[0]);
44         }
45
46         if (x_k.array().isNaN().sum() > 0 && tid == 13) {
47             printf("WARNING:: x_k NAN detected! on timestep: %i, and rollout %i\n", k, tid);
48             printf("x(%i)=%f\n", k, x_k[0]);
49         }
50     }
51
52     __syncthreads();
53 }
```

The screenshot shows the Microsoft Visual Studio IDE interface with the GATE\_dubins.cu file open in the editor. The code implements a CUDA kernel for a quadcopter simulation. It includes loops for state trajectories, dynamics integration, and NaN detection. The code is annotated with line numbers and color-coded syntax highlighting for C++ and CUDA. The Visual Studio interface includes a toolbar, a menu bar, and various tool windows on the right side.



# Runtime Comparisons

# Case Study: Quadcopter





# Conclusions



# Conclusions



GATE enables:

- Plug-and-play abilities
- Fast performance

Future work:

- Variable timestep integration scheme
- Multiple GPU usage

Contacts:

- [mgandhi@sandia.gov](mailto:mgandhi@sandia.gov)
- [rschlos@sandia.gov](mailto:rschlos@sandia.gov)

