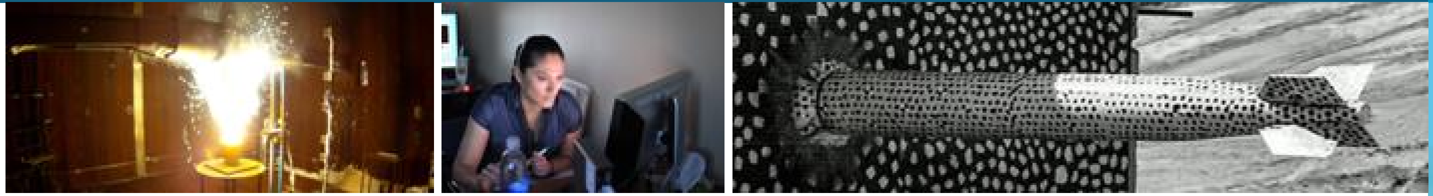
This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2021-14301C

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

For audio, see https://youtu.be/ay7ti_OfPmk

U.S. DEPARTMENT OF ENERGY NNSA National Nuclear Security Administration

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.

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.

Introduction

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 Overview

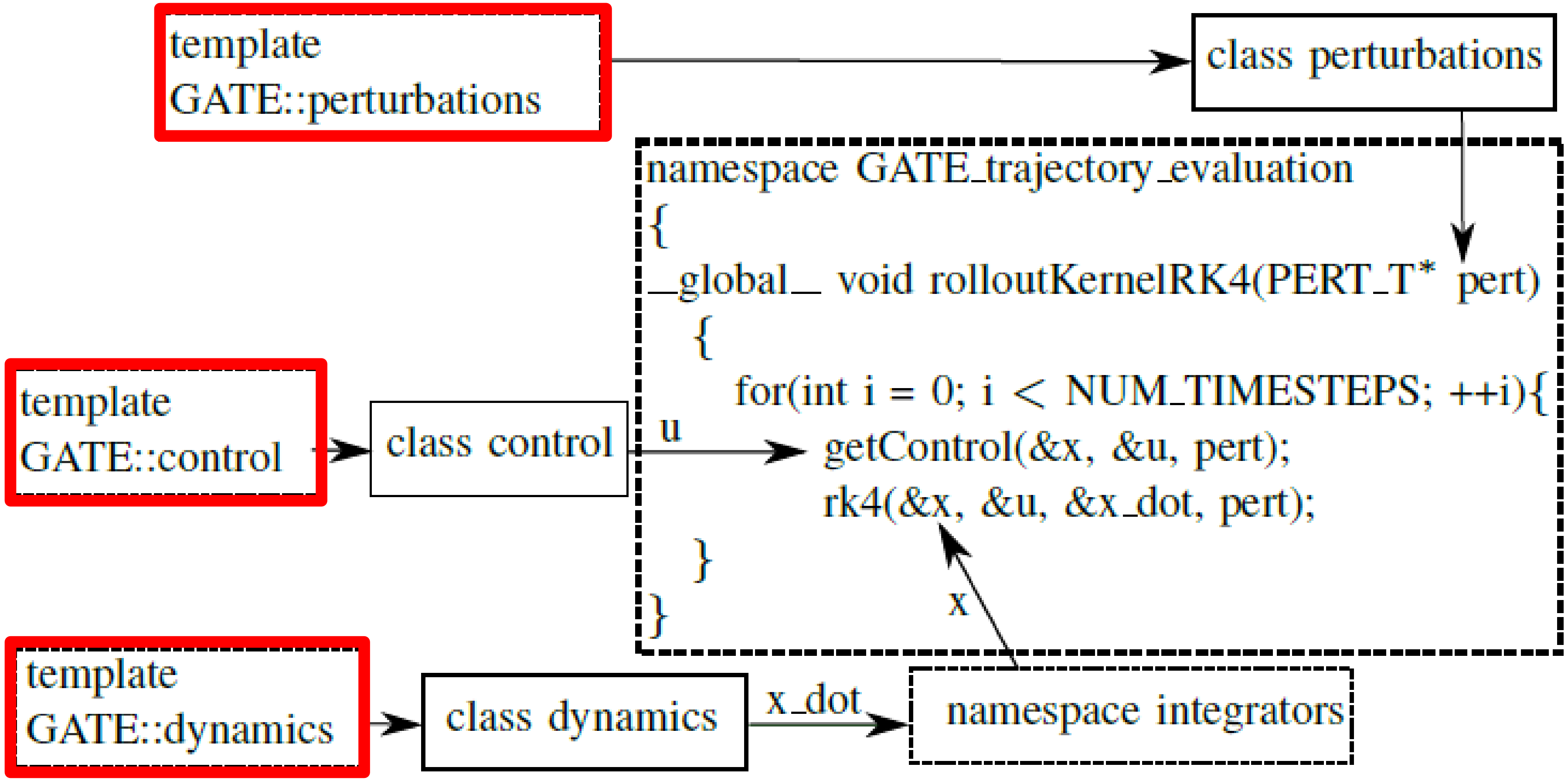
CUDA (Compute Unified Device Architecture) is a programming interface based on C which allows users to take advantage of massive parallelism offered by GPU's for general-purpose programming.

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

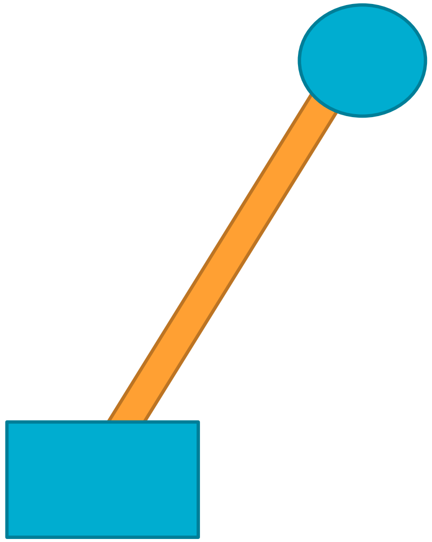
GATE Architecture: Design Philosophy

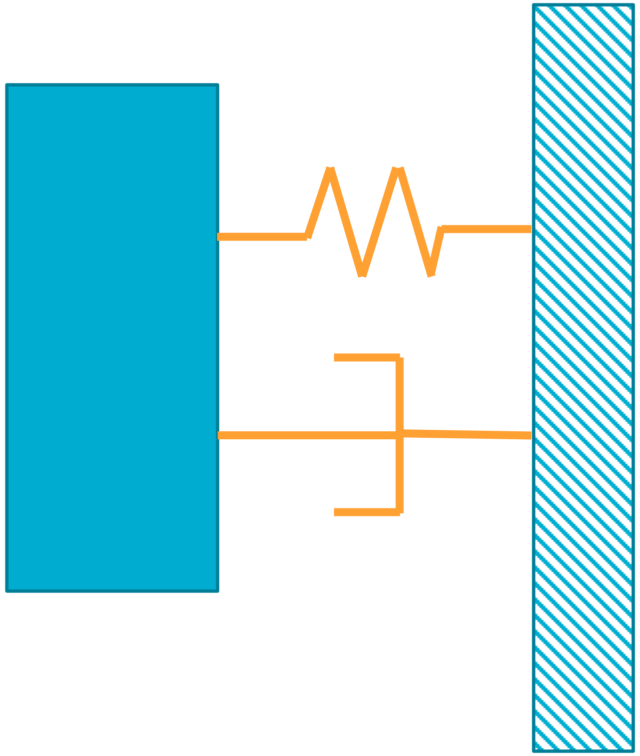


```
namespace GATE_trajectory_evaluation
{
__global__ void rolloutKernelRK4(PERT_T* pert)
  {
    for(int i = 0; i < NUM_TIMESTEPS; ++i){
      getControl(&x, &u, pert);
      rk4(&x, &u, &x_dot, pert);
  }
}
```

GATE Architecture: Core Classes

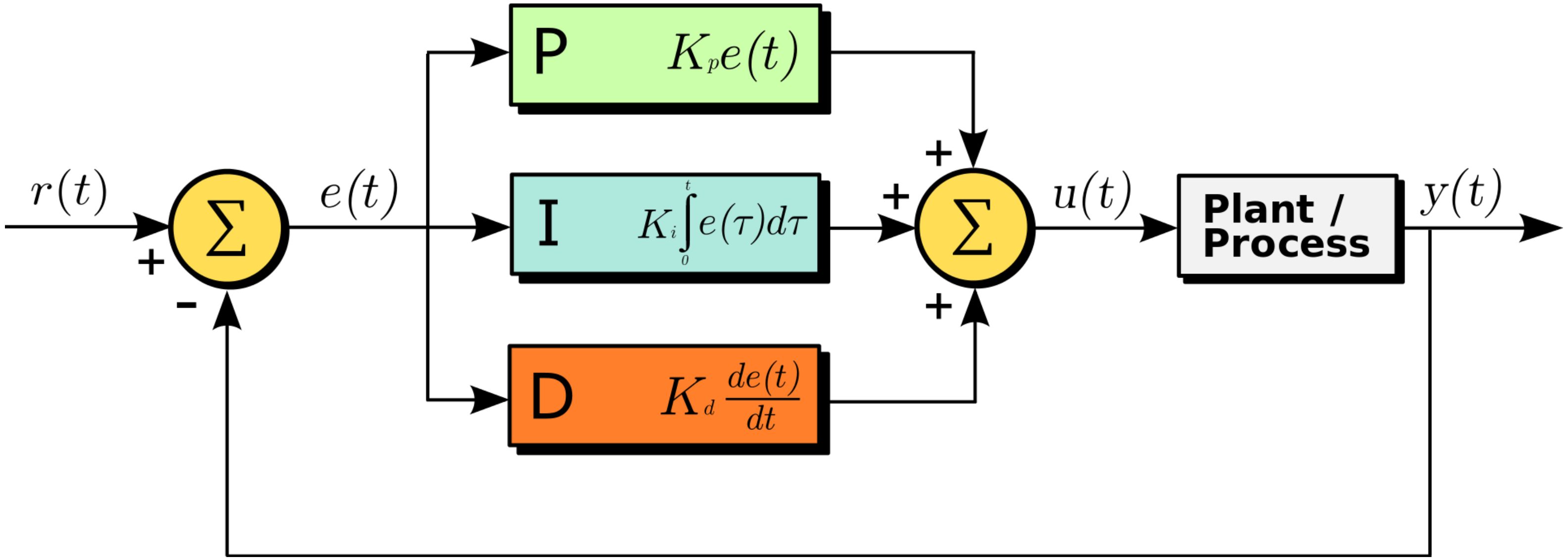
- **Dynamics:**
  - Implements state derivative computation given a control input and set of perturbations.
  - Can be modified to represent multi-phase, hybrid systems.





GATE Architecture: Core Classes

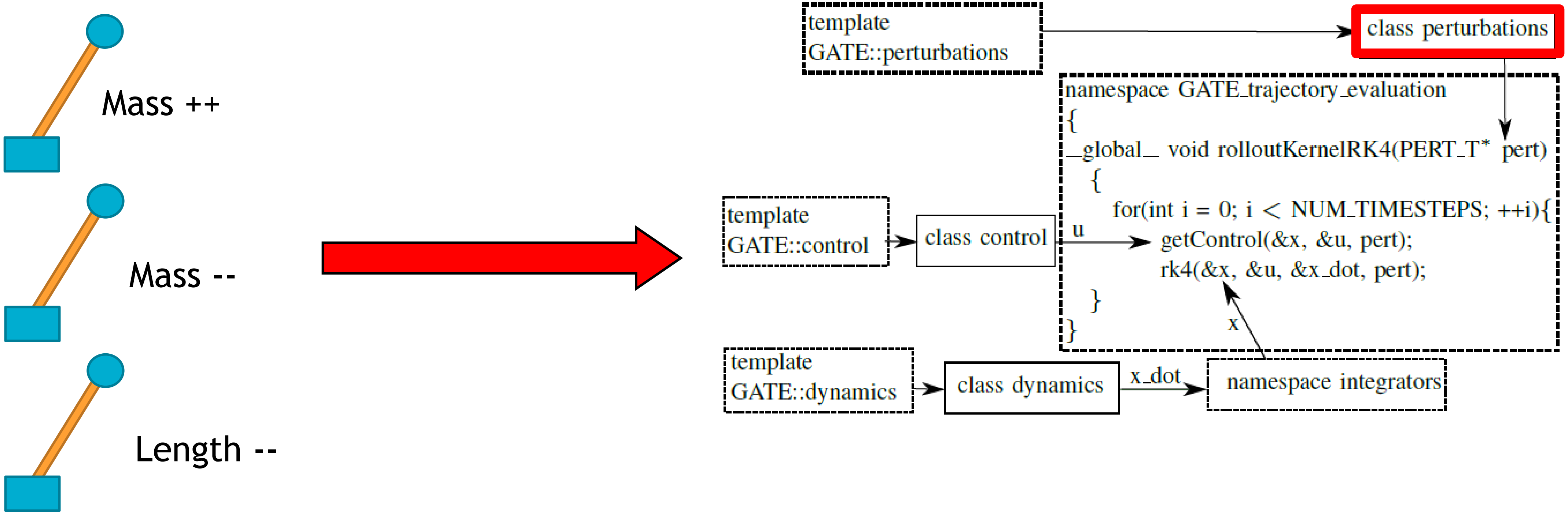
- **Controllers:**
  - Implements various control strategies designed by the end user.
  - Support for per-rollout parameters allows for adaptation of the controller based on the simulation.



Source: Wikipedia Commons: https://upload.wikimedia.org/wikipedia/commons/thumb/4/43/PID_en.svg/1920px-PID_en.svg.png

GATE Architecture: Core Classes

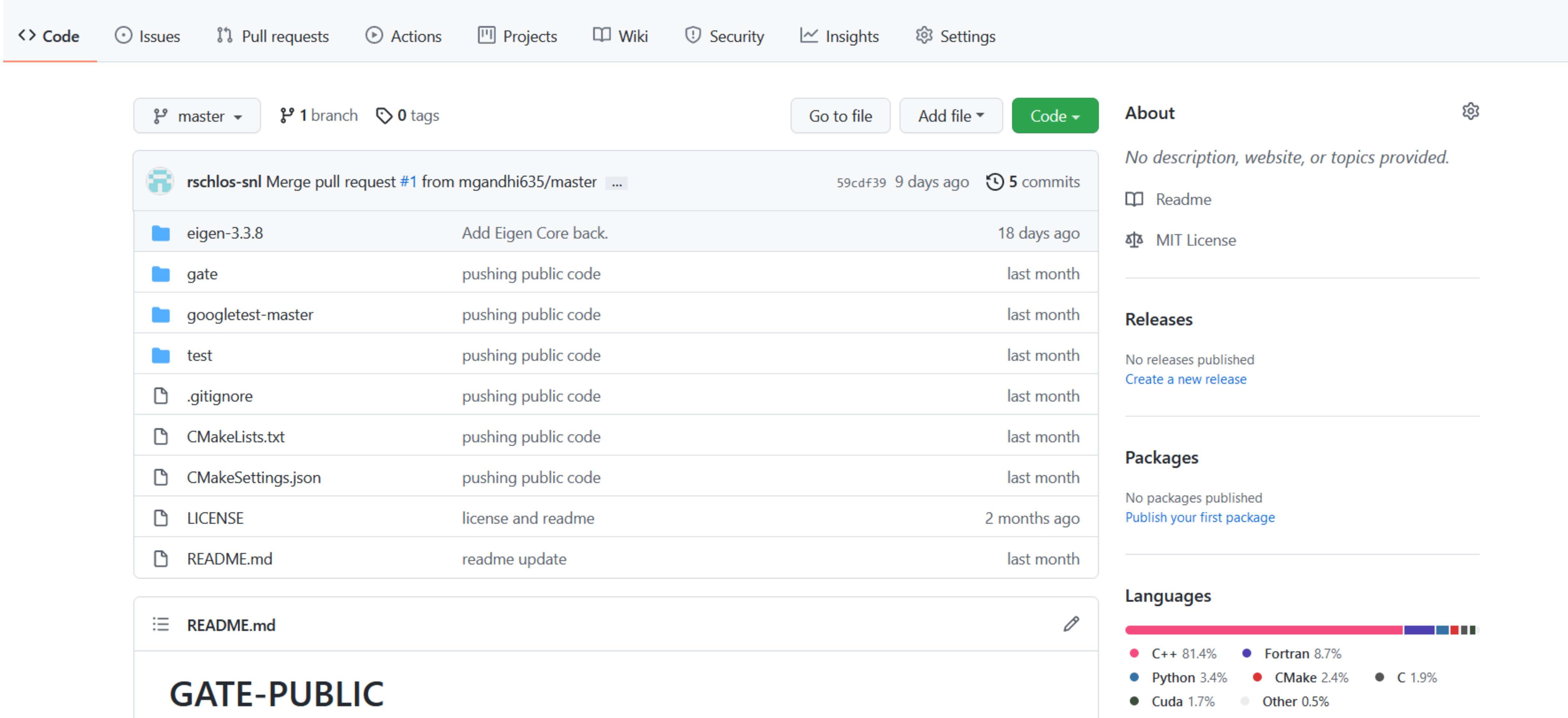
- **Perturbation**
  - Implements perturbations in the initial condition and in the control channel.
  - Support for variation in parameters in both time horizon and number of rollouts.



```
namespace GATE_trajectory_evaluation
{
__global__ void rolloutKernelRK4(PERT_T* pert)
  {
    for(int i = 0; i < NUM_TIMESTEPS; ++i){
        getControl(&x, &u, pert);
        rk4(&x, &u, &x_dot, pert);
  }
}
```

Implementation Details

Getting Started



rschlos-snl 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

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

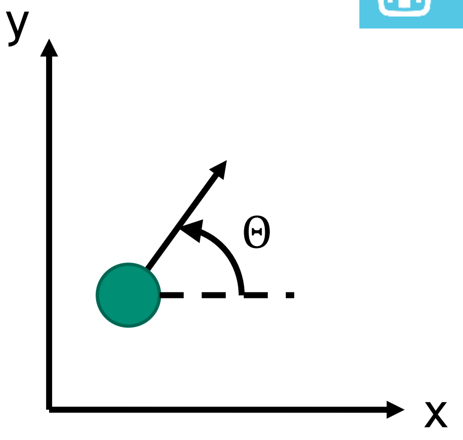
Languages

● C++ 81.4% ● Fortran 8.7% ● Python 3.4% ● CMake 2.4% ● C 1.9% ● Cuda 1.7% ● Other 0.5%

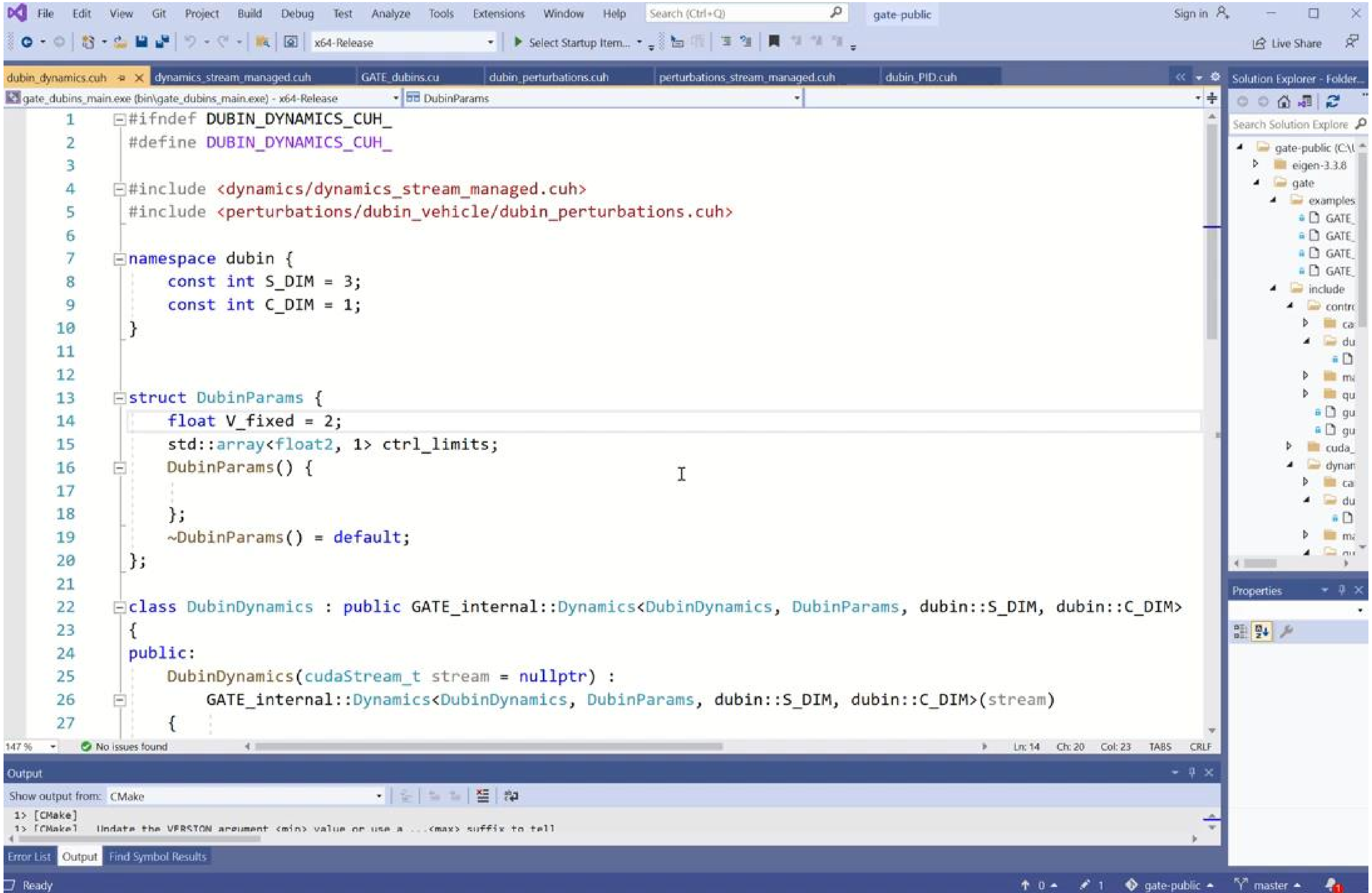
https://github.com/sandialabs/gate-public

Example: Dubins Vehicle

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



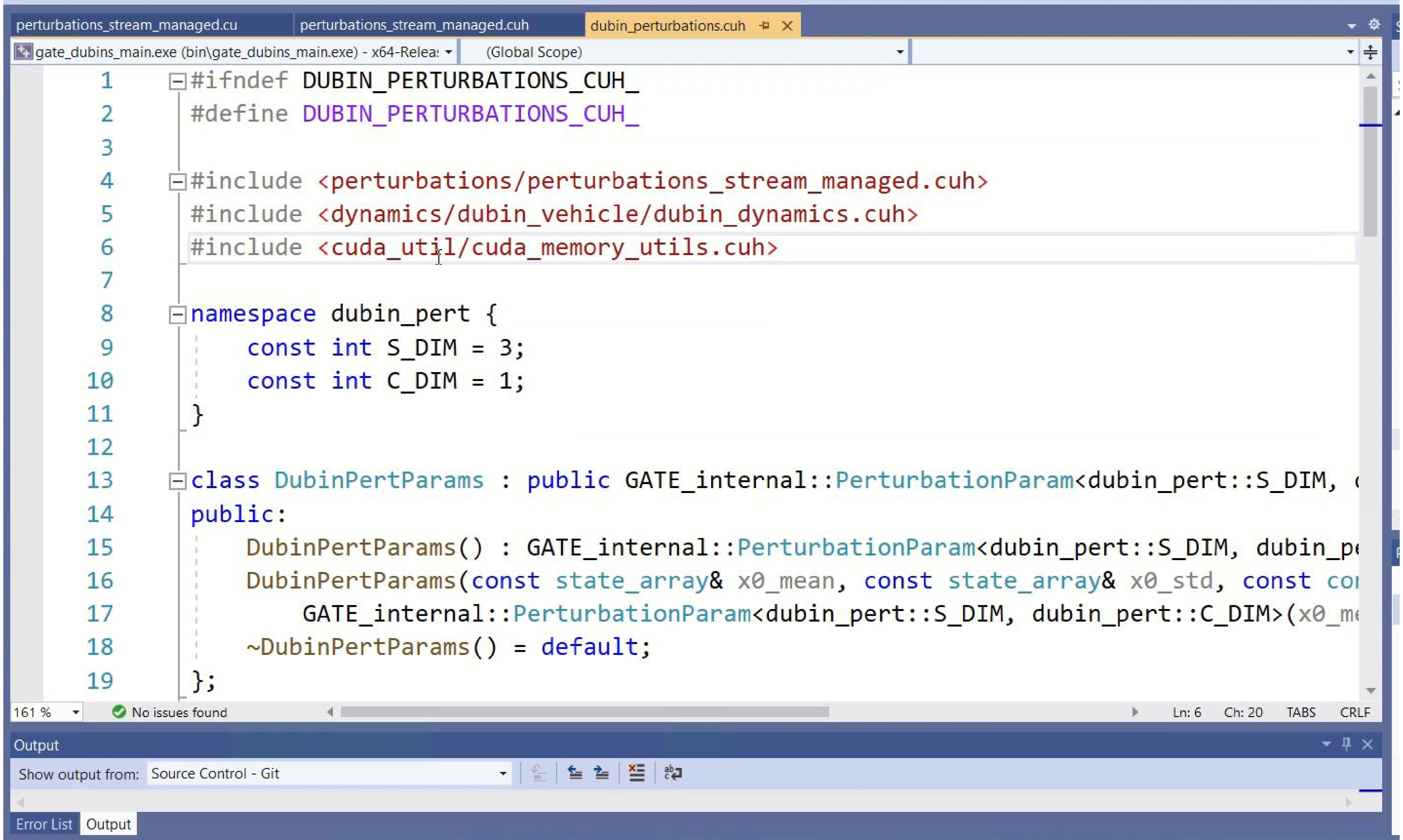
Dubins Dynamics Class



```
#ifndef DUBIN_DYNAMICS_CUH_
#define DUBIN_DYNAMICS_CUH_

#include <dynamics/dynamics_stream_managed.cuh>
#include <perturbations/dubin_vehicle/dubin_perturbations.cuh>

namespace dubin {
    const int S_DIM = 3;
    const int C_DIM = 1;
}


struct DubinParams {
    float V_fixed = 2;
    std::array<float2, 1> ctrl_limits;
    DubinParams() {

    };
    ~DubinParams() = default;
};

class DubinDynamics : public GATE_internal::Dynamics<DubinDynamics, DubinParams, dubin::S_DIM, dubin::C_DIM>
{
public:
    DubinDynamics(cudaStream_t stream = nullptr) :
        GATE_internal::Dynamics<DubinDynamics, DubinParams, dubin::S_DIM, dubin::C_DIM>(stream)
    {
```

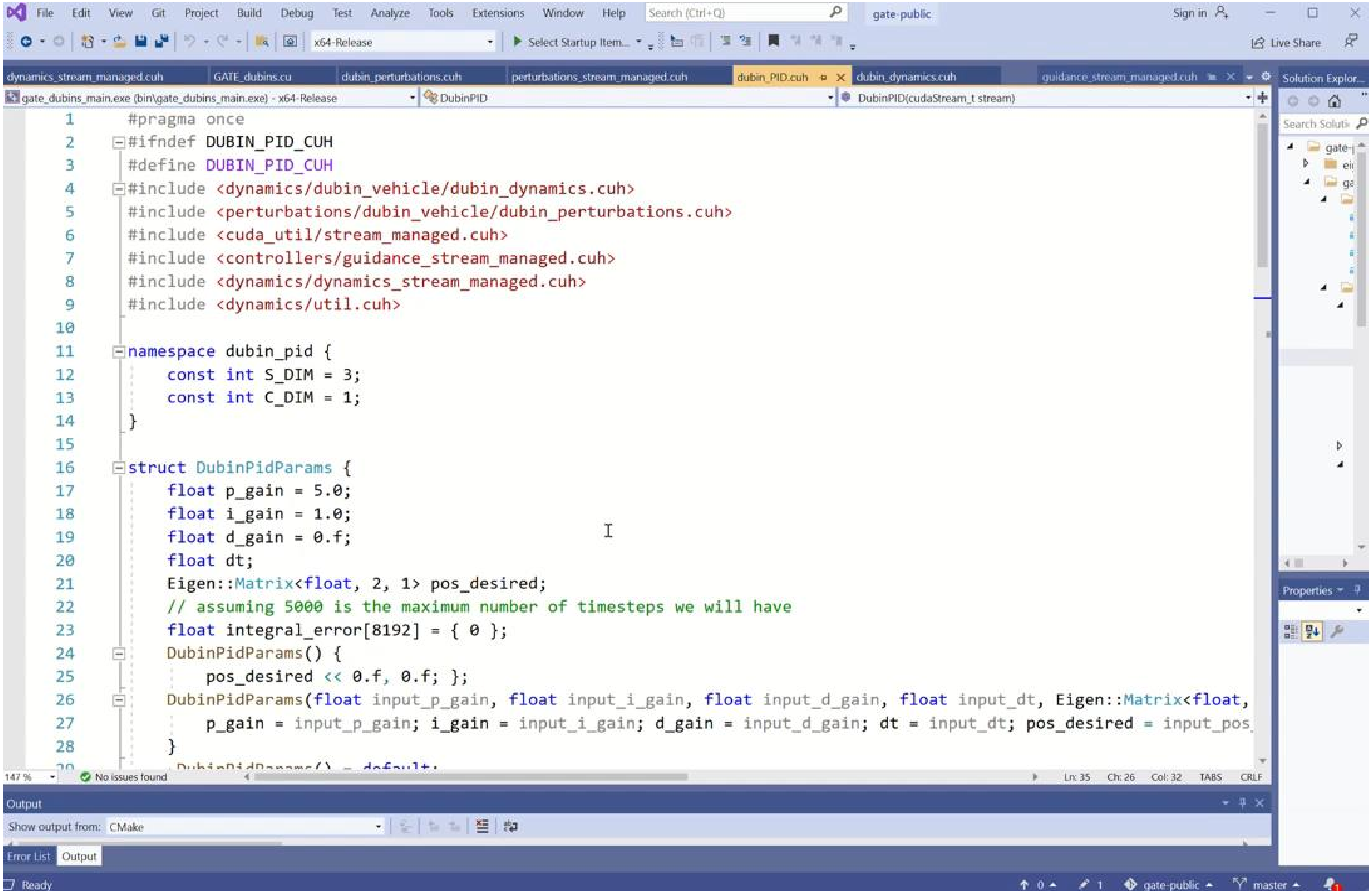
```
1> [CMake]
1> [CMake]   Update the VERSION argument <min> value or use a ...<max> suffix to tell
```

Dubins Perturbations Class



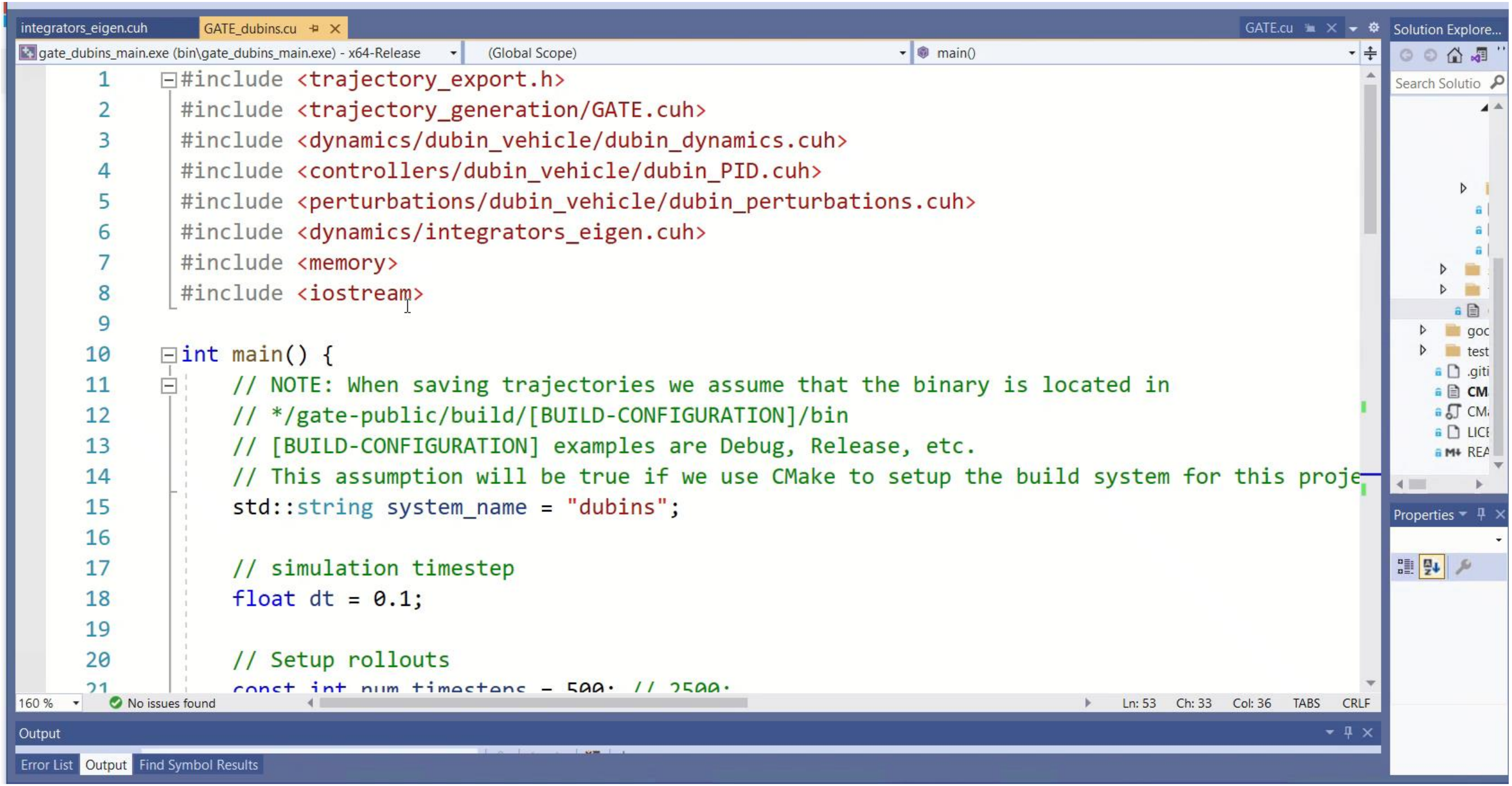
```
#ifndef DUBIN_PERTURBATIONS_CUH_
#define DUBIN_PERTURBATIONS_CUH_

#include <perturbations/perturbations_stream_managed.cuh>
#include <dynamics/dubin_vehicle/dubin_dynamics.cuh>
#include <cuda_util/cuda_memory_utils.cuh>

namespace dubin_pert {
    const int S_DIM = 3;
    const int C_DIM = 1;
}

class DubinPertParams : public GATE_internal::PerturbationParam<dubin_pert::S_DIM,
public:
    DubinPertParams() : GATE_internal::PerturbationParam<dubin_pert::S_DIM, dubin_pe
    DubinPertParams(const state_array& x0_mean, const state_array& x0_std, const co
        GATE_internal::PerturbationParam<dubin_pert::S_DIM, dubin_pert::C_DIM>(x0_me
    ~DubinPertParams() = default;
};
```

Dubins Control Class



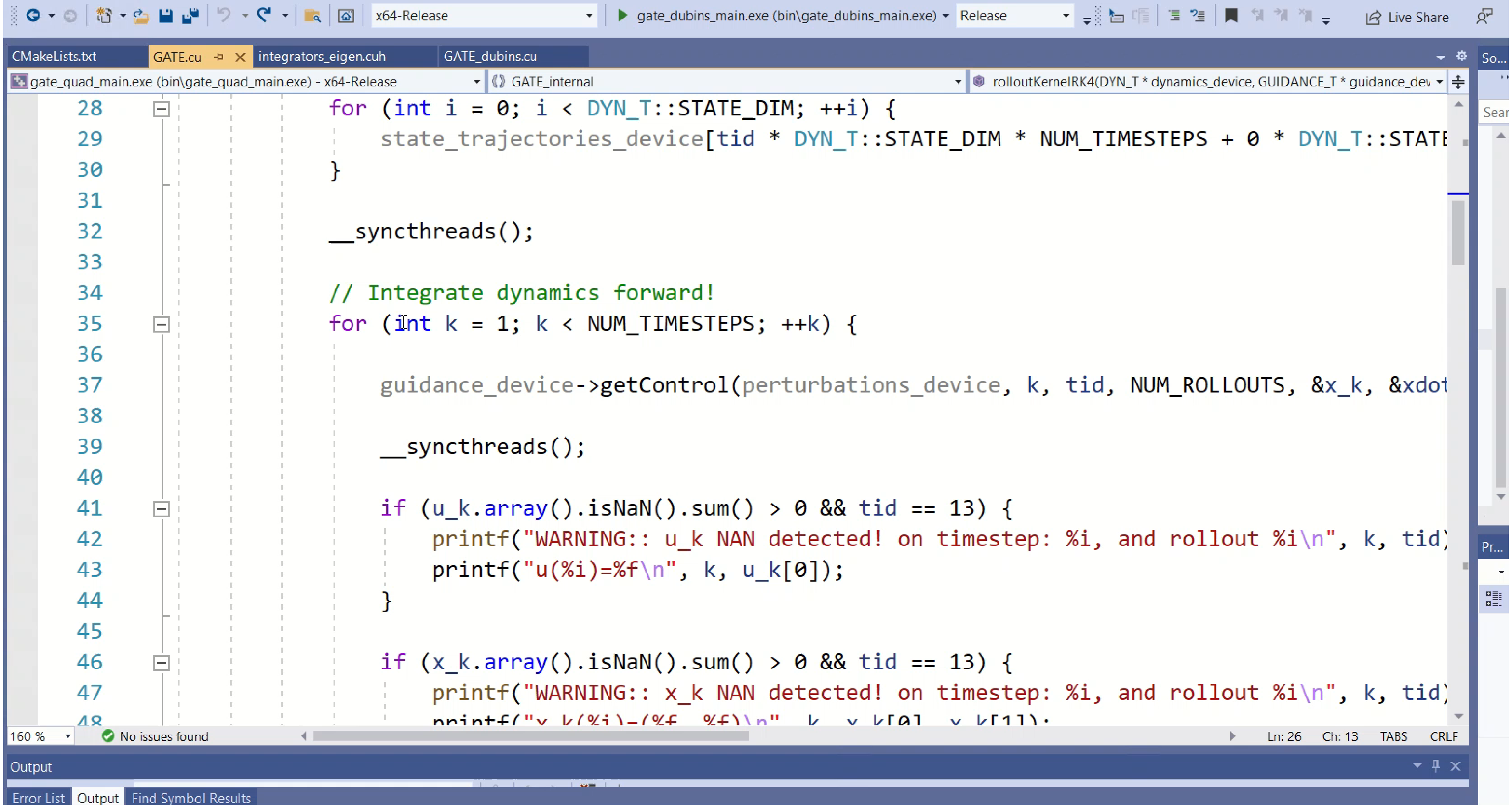
```
#pragma once
#ifndef DUBIN_PID_CUH
#define DUBIN_PID_CUH
#include <dynamics/dubin_vehicle/dubin_dynamics.cuh>
#include <perturbations/dubin_vehicle/dubin_perturbations.cuh>
#include <cuda_util/stream_managed.cuh>
#include <controllers/guidance_stream_managed.cuh>
#include <dynamics/dynamics_stream_managed.cuh>
#include <dynamics/util.cuh>

namespace dubin_pid {
    const int S_DIM = 3;
    const int C_DIM = 1;
}

struct DubinPidParams {
    float p_gain = 5.0;
    float i_gain = 1.0;
    float d_gain = 0.f;
    float dt;
    Eigen::Matrix<float, 2, 1> pos_desired;
    // assuming 5000 is the maximum number of timesteps we will have
    float integral_error[8192] = { 0 };
    DubinPidParams() {
        pos_desired << 0.f, 0.f; };
    DubinPidParams(float input_p_gain, float input_i_gain, float input_d_gain, float input_dt, Eigen::Matrix<float,
        p_gain = input_p_gain; i_gain = input_i_gain; d_gain = input_d_gain; dt = input_dt; pos_desired = input_pos
    }
    DubinPidParams() = default;
```

Running the Simulation



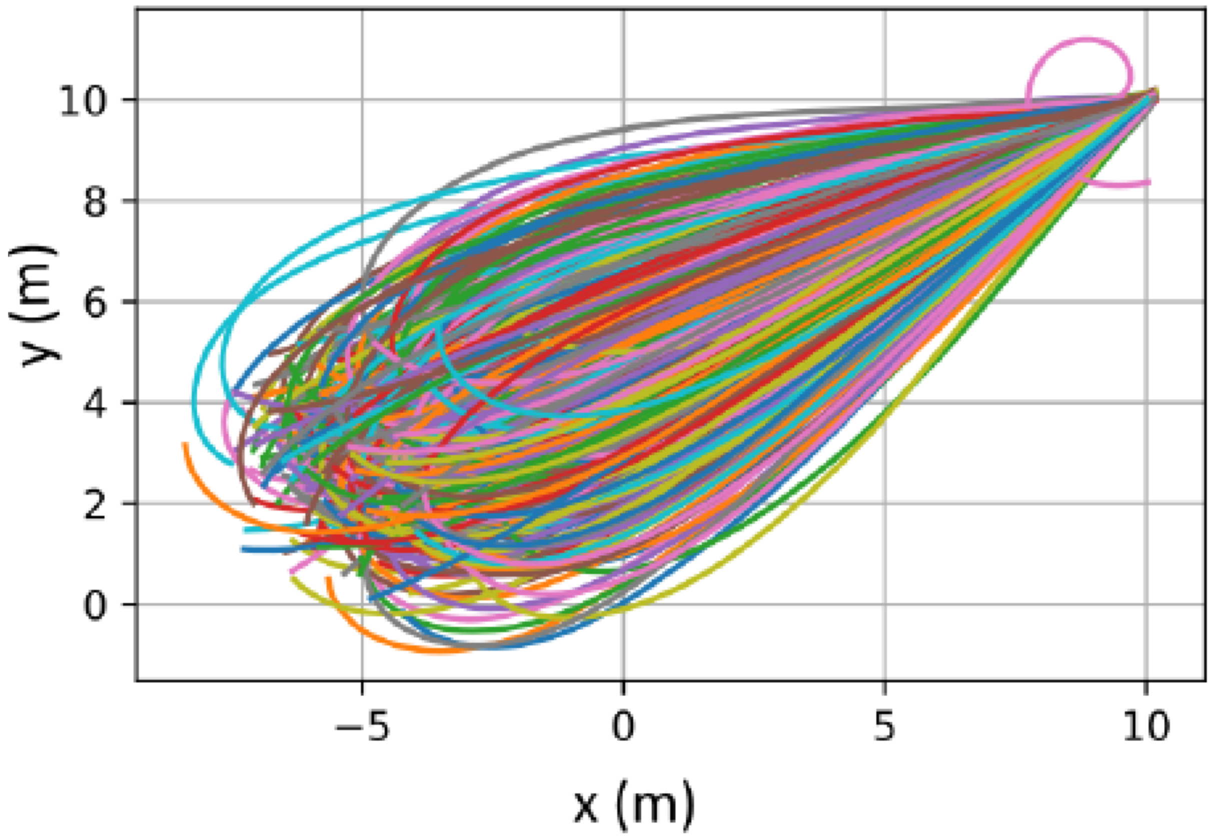
```
#include <trajectory_export.h>
#include <trajectory_generation/GATE.cuh>
#include <dynamics/dubin_vehicle/dubin_dynamics.cuh>
#include <controllers/dubin_vehicle/dubin_PID.cuh>
#include <perturbations/dubin_vehicle/dubin_perturbations.cuh>
#include <dynamics/integrators_eigen.cuh>
#include <memory>
#include <iostream>

int main() {
    // NOTE: When saving trajectories we assume that the binary is located in
    // */gate-public/build/[BUILD-CONFIGURATION]/bin
    // [BUILD-CONFIGURATION] examples are Debug, Release, etc.
    // This assumption will be true if we use CMake to setup the build system for this proje
    std::string system_name = "dubins";

    // simulation timestep
    float dt = 0.1;

    // Setup rollouts
    const int num_timesteps = 500; // 2500;
```

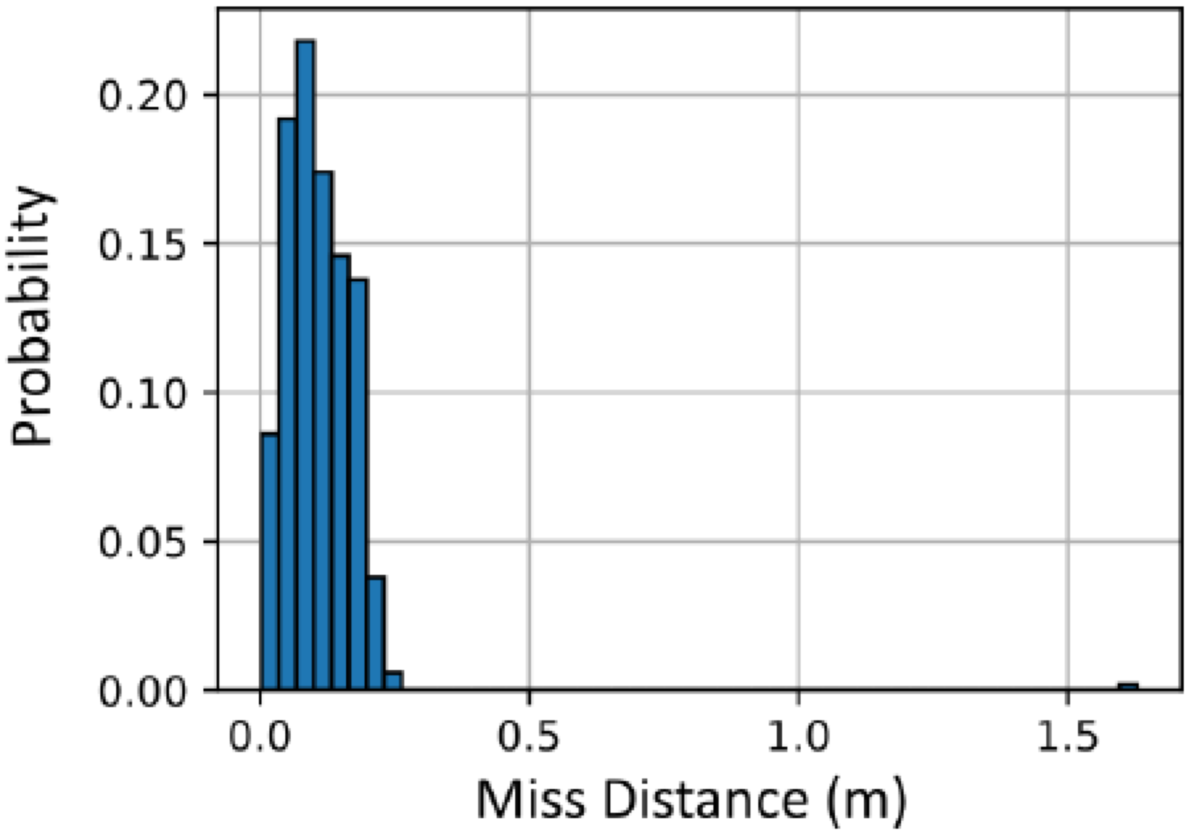
Running the Simulation



```
for (int i = 0; i < DYN_T::STATE_DIM; ++i) {
    state_trajectories_device[tid * DYN_T::STATE_DIM * NUM_TIMESTEPS + 0 * DYN_T::STATE
}

__syncthreads();

// Integrate dynamics forward!
for (int k = 1; k < NUM_TIMESTEPS; ++k) {

    guidance_device->getControl(perturbations_device, k, tid, NUM_ROLLOUTS, &x_k, &xdot

    __syncthreads();

    if (u_k.array().isNaN().sum() > 0 && tid == 13) {
        printf("WARNING:: u_k NAN detected! on timestep: %i, and rollout %i\n", k, tid)
        printf("u(%i)=%f\n", k, u_k[0]);
    }

    if (x_k.array().isNaN().sum() > 0 && tid == 13) {
        printf("WARNING:: x_k NAN detected! on timestep: %i, and rollout %i\n", k, tid)
        printf("x_k(%i)=(%f, %f)\n", k, x_k[0], x_k[1]);
```

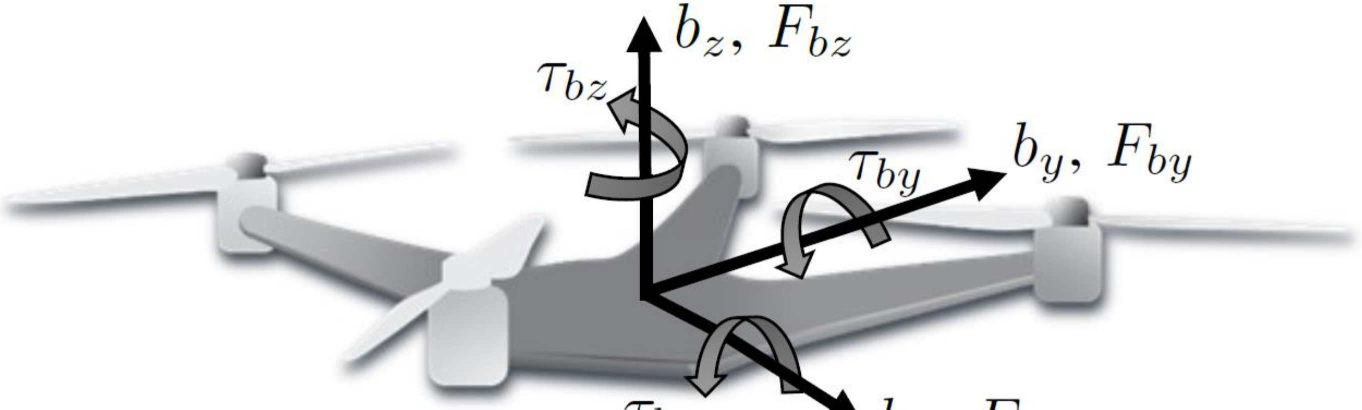
Results

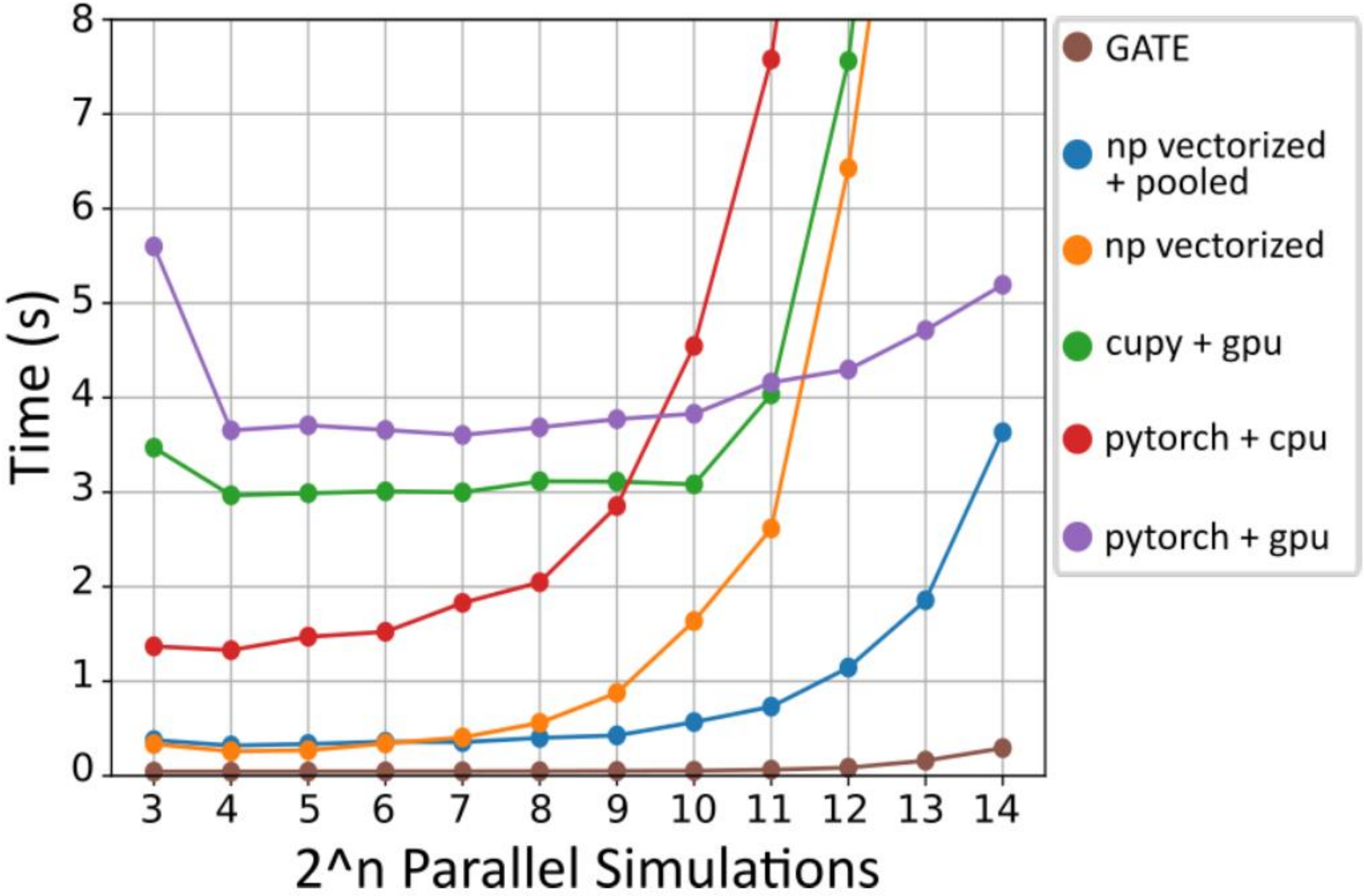

y (m)
10
8
6
4
2
0
−5
0
5
10
x (m)

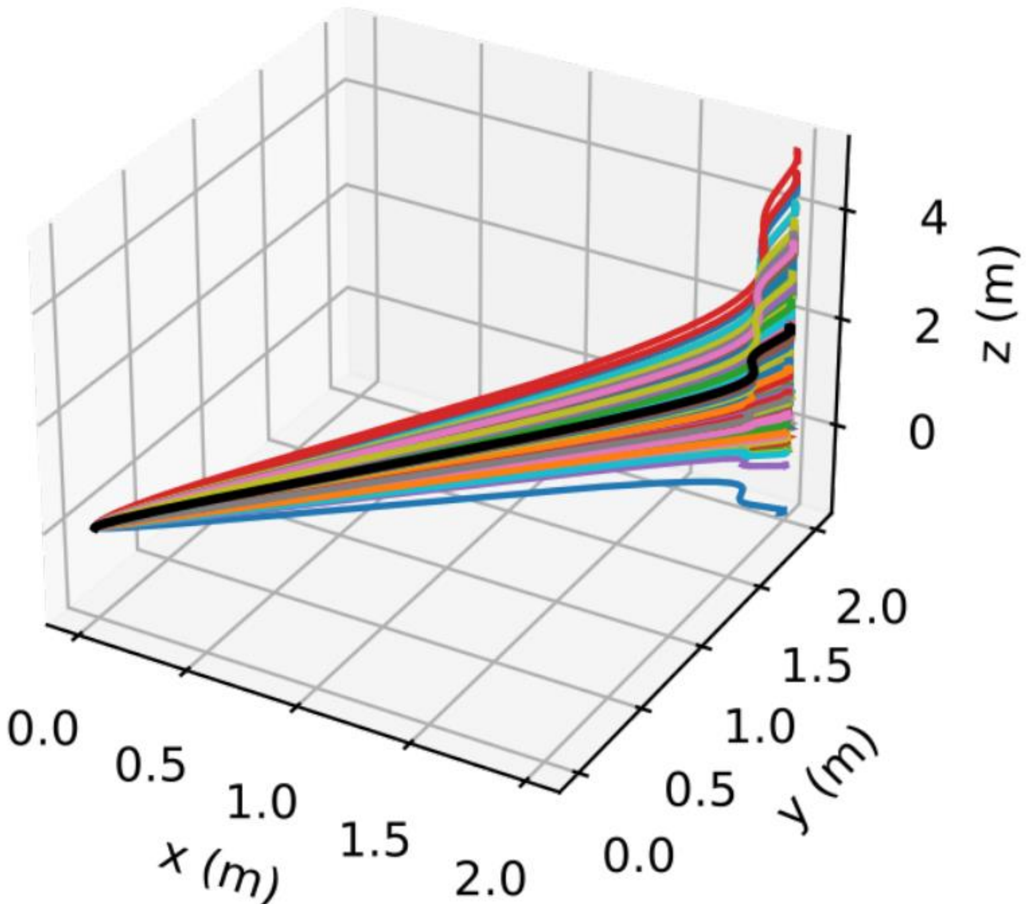

Probability
0.20
0.15
0.10
0.05
0.00
0.0
0.5
1.0
1.5
Miss Distance (m)

Runtime Comparisons

Case Study: Quadcopter







Time to simulate 15 seconds (1500 timesteps, dt = 0.01 s)

Conclusions

Conclusions

GATE enables:

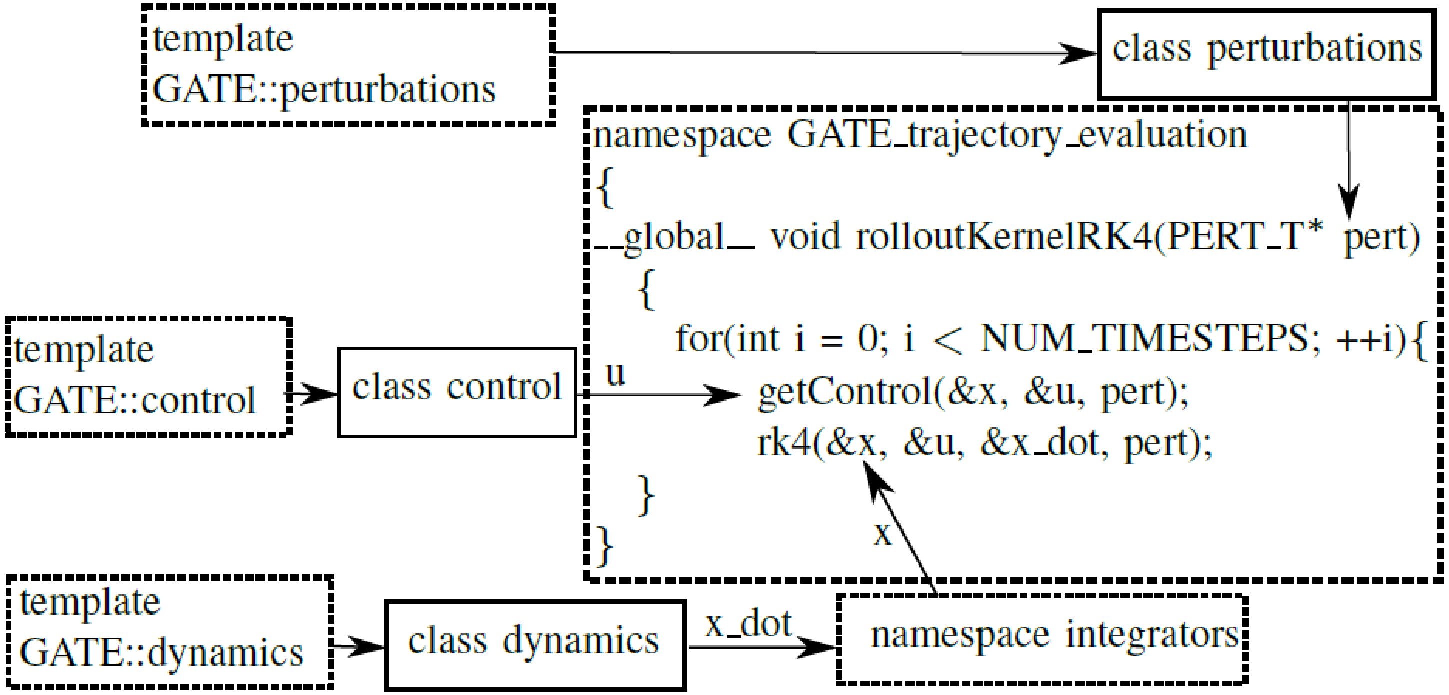
- Plug-and-play abilities
- Fast performance

Future work:

- Variable timestep integration scheme
- Multiple GPU usage

Contacts:

- mgandhi@sandia.gov
- rschlos@sandia.gov



```
namespace GATE_trajectory_evaluation
{
__global__ void rolloutKernelRK4(PERT_T* pert)
  {
    for(int i = 0; i < NUM_TIMESTEPS; ++i){
      getControl(&x, &u, pert);
      rk4(&x, &u, &x_dot, pert);
  }
}
```