

## Embedded Systems Lab Task 1

### PID Control of Inverted Pendulum



<b>1 Getting Started</b>	<b>1</b>
1.1 Table of Contents	1
1.2 Introduction	1
1.3 Features	1
1.4 Dependencies	1
1.5 Building the Project	2
1.6 Usage	2
<b>2 Todo List</b>	<b>3</b>
<b>3 Hierarchical Index</b>	<b>5</b>
3.1 Class Hierarchy	5
<b>4 Class Index</b>	<b>7</b>
4.1 Class List	7
<b>5 File Index</b>	<b>9</b>
5.1 File List	9
<b>6 Class Documentation</b>	<b>11</b>
6.1 Cart Struct Reference	11
6.1.1 Detailed Description	11
6.1.2 Member Data Documentation	11
6.1.2.1 l	11
6.1.2.2 len	12
6.1.2.3 M	12
6.1.2.4 m	12
6.2 CommServer Class Reference	12
6.2.1 Detailed Description	12
6.2.2 Constructor & Destructor Documentation	12
6.2.2.1 CommServer()	12
6.2.3 Member Function Documentation	13
6.2.3.1 start_server()	13
6.3 Controller Class Reference	13
6.4 PIDController Class Reference	13
6.4.1 Detailed Description	14
6.4.2 Constructor & Destructor Documentation	14
6.4.2.1 PIDController()	14
6.4.3 Member Function Documentation	14
6.4.3.1 output()	14
6.4.3.2 setClamp()	15
6.4.3.3 update_params()	15
6.4.4 Member Data Documentation	15
6.4.4.1 kd	16

6.4.4.2 ki	16
6.4.4.3 kp	16
6.5 SimParams Struct Reference	16
6.5.1 Detailed Description	17
6.5.2 Member Data Documentation	17
6.5.2.1 delay	17
6.5.2.2 delta_t	17
6.5.2.3 g	17
6.5.2.4 jitter	17
6.5.2.5 ref_angle	17
6.5.2.6 simulation_time	18
6.6 Simulator Class Reference	18
6.6.1 Detailed Description	19
6.6.2 Constructor & Destructor Documentation	19
6.6.2.1 Simulator() [1/2]	20
6.6.2.2 Simulator() [2/2]	20
6.6.3 Member Function Documentation	20
6.6.3.1 reset_simulator()	20
6.6.3.2 run_simulator()	20
6.6.3.3 update_params()	21
6.6.4 Member Data Documentation	21
6.6.4.1 a	21
6.6.4.2 A	21
6.6.4.3 B	21
6.6.4.4 b	22
6.6.4.5 buffer_size	22
6.6.4.6 C	22
6.6.4.7 c	22
6.6.4.8 c_ml	22
6.6.4.9 E	22
6.6.4.10 error	23
6.6.4.11 F	23
6.6.4.12 g_pause	23
6.6.4.13 g_pause_cv	23
6.6.4.14 g_pause_mutex	23
6.6.4.15 g_reset	23
6.6.4.16 g_start	24
6.6.4.17 g_start_cv	24
6.6.4.18 g_start_mutex	24
6.6.4.19 i	24
6.6.4.20 m_cart	24
6.6.4.21 m_controller	24

6.6.4.22 m_params . . . . .	25
6.6.4.23 T . . . . .	25
6.6.4.24 theta . . . . .	25
6.6.4.25 theta_dot . . . . .	25
6.6.4.26 theta_dot_dot . . . . .	25
6.6.4.27 x . . . . .	26
6.6.4.28 x_dot . . . . .	26
6.6.4.29 x_dot_dot . . . . .	26
<b>7 File Documentation</b>	<b>27</b>
7.1 es_lab/task_1/simulator/include/controller.h File Reference . . . . .	27
7.1.1 Detailed Description . . . . .	27
7.2 controller.h . . . . .	28
7.3 es_lab/task_1/simulator/include/server.h File Reference . . . . .	28
7.3.1 Detailed Description . . . . .	29
7.3.2 Typedef Documentation . . . . .	29
7.3.2.1 json . . . . .	29
7.3.2.2 tcp . . . . .	29
7.4 server.h . . . . .	29
7.5 es_lab/task_1/simulator/include/simulator.h File Reference . . . . .	30
7.5.1 Detailed Description . . . . .	30
7.6 simulator.h . . . . .	31
7.7 es_lab/task_1/simulator/README.md File Reference . . . . .	32
7.8 es_lab/task_1/simulator/src/controller.cpp File Reference . . . . .	32
7.8.1 Detailed Description . . . . .	32
7.9 es_lab/task_1/simulator/src/main.cpp File Reference . . . . .	32
7.9.1 Detailed Description . . . . .	33
7.9.2 Function Documentation . . . . .	33
7.9.2.1 main() . . . . .	33
7.10 es_lab/task_1/simulator/src/server.cpp File Reference . . . . .	33
7.10.1 Detailed Description . . . . .	34
7.11 es_lab/task_1/simulator/src/simulator.cpp File Reference . . . . .	34
7.11.1 Detailed Description . . . . .	34
<b>Index</b>	<b>35</b>



# Chapter 1

## Getting Started

### 1.1 Table of Contents

- Introduction
- Features
- Building the Project
- Usage

### 1.2 Introduction

A C++ simulator that implements PID control for an inverted pendulum system. It provides a simulation environment where the user can test and evaluate different PID controller parameters to stabilize the inverted pendulum.

The simulator also includes an HTTP server component that allows external control and monitoring of the simulation parameters via frontend requests. This enables users to interact with the simulation in real-time through a web-based interface.

### 1.3 Features

- Simulates PID control of an inverted pendulum system.
- Dynamically adjustable PID controller parameters ( $k_p$ ,  $k_d$ ,  $k_i$ ).
- Real-time visualization and monitoring of simulation.
- HTTP server for remote control and monitoring via web interface.

### 1.4 Dependencies

- CMake - For building the project .
- Boost - For HTTP server, release version can be downloaded from [boost website](#)
- nlohmann/json - json.hpp file can be downloaded from [github](#)

## 1.5 Building the Project

To install the simulator, follow these steps:

1. Clone the repository to your local machine:  
`git clone git@github.com:linem-davton/es-lab-task1.git`
2. Navigate to the project directory:  
`cd es-lab-task1`
3. Create a build and libs directory in the project directory  
`mkdir build`  
`mkdir libs`  
`mkdir llibs/nlohmann`
4. Rename the unzipped boost directory to boost in the *es-lab-task1/libs* directory
5. Place the json.hpp file in the *es-lab-task1/libs/nlohmann* directory
6. Navigate to the build directory:  
`cd build`
7. Configure the project with CMake:  
`cmake ..`
8. Build the project:  
`make`

If you encounter any issues during the build process, please check the dependencies and ensure that they are correctly installed and configured. Make sure CMakeLists.txt is correctly configured to include the required libraries.

## 1.6 Usage

To use the simulator, follow these steps:

1. Navigate to the build directory where the project was built:  
`cd es-lab-task1/build`
2. Run the simulator binary:  
`./simulator`
3. Download the frontend code from [github](#)
4. Follow the instructions in the frontend README to start the frontend server.
5. Open a web browser and navigate to the frontend server address (default: [http://localhost↵:3000](http://localhost:3000)).
6. Use the web interface to control and monitor the simulation parameters. You can start, stop, and adjust the simulation settings as needed.
7. Interact with the simulator via the web interface to observe the behavior of the inverted pendulum system under different control conditions.

By following these steps, you can run the simulator and control it using the web interface provided by the frontend server.



## Chapter 2

### Todo List

**Member `PIDController::output` (double error)**

Implement the PID controller output calculation

**Member `PIDController::setClamp` (double max, double min)**

Implement setClamp for setting the output limits

**Member `PIDController::update_params` (double kp, double kd, double ki)**

Implement the update\_params function for PID controller

**Member `Simulator::run_simulator` ()**

Implement delay and jitter by changing delay index based on [SimParams.delay](#) and [SimParams.jitter](#)

Handle case when delay index is negative, wrap around to end of circular buffer

Make sure delay index is within bounds of buffer size

**Member `Simulator::update_params` (double ref, int delay, int jitter)**

Implement update\_params function to update simulation parameters



## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Cart . . . . .	11
CommServer . . . . .	12
Controller . . . . .	13
PIDController . . . . .	13
SimParams . . . . .	16
Simulator . . . . .	18



## Chapter 4

# Class Index

### 4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">Cart</a>	Struct containing parameters for the cart . . . . .	11
<a href="#">CommServer</a>	Class for managing communication frontend and simulation backend . . . . .	12
<a href="#">Controller</a>	Interface for controllers used in the inverted pendulum simulation . . . . .	13
<a href="#">PIDController</a>	Implementation of a PID (Proportional-Integral-Derivative) controller . . . . .	13
<a href="#">SimParams</a>	Struct containing parameters for simulation . . . . .	16
<a href="#">Simulator</a>	<a href="#">Simulator</a> class for simulating the inverted pendulum . . . . .	18



## Chapter 5

# File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

es_lab/task_1/simulator/include/controller.h	
Header file for the <a href="#">Controller</a> interface and <a href="#">PIDController</a> class . . . . .	27
es_lab/task_1/simulator/include/server.h	
Header file for the <a href="#">CommServer</a> class . . . . .	28
es_lab/task_1/simulator/include/simulator.h	
Header file for <a href="#">Simulator</a> class, <a href="#">SimParams</a> and <a href="#">Cart</a> . . . . .	30
es_lab/task_1/simulator/src/controller.cpp	
Implementation file for the <a href="#">PIDController</a> class . . . . .	32
es_lab/task_1/simulator/src/main.cpp	
Main entry point for the inverted pendulum simulation . . . . .	32
es_lab/task_1/simulator/src/server.cpp	
Implementation file for the <a href="#">CommServer</a> class . . . . .	33
es_lab/task_1/simulator/src/simulator.cpp	
Implementation file for the <a href="#">Simulator</a> class . . . . .	34





## Chapter 6

# Class Documentation

### 6.1 Cart Struct Reference

Struct containing parameters for the cart.

```
#include <es_lab/task_1/simulator/include/simulator.h>
```

#### Public Attributes

- double `M` = 5  
*Mass of cart.*
- double `m` = 0.5  
*Mass of pendulum.*
- double `len` = 1  
*Pendulum center of mass to pivot point.*
- double `I` = `m * len * len`  
*Moment of inertia of pendulum.*

#### 6.1.1 Detailed Description

Struct containing parameters for the cart.

#### 6.1.2 Member Data Documentation

##### 6.1.2.1 `I`

```
double Cart::I = m * len * len
```

Moment of inertia of pendulum.

### 6.1.2.2 len

```
double Cart::len = 1
```

Pendulum center of mass to pivot point.

### 6.1.2.3 M

```
double Cart::M = 5
```

Mass of cart.

### 6.1.2.4 m

```
double Cart::m = 0.5
```

Mass of pendulum.

The documentation for this struct was generated from the following file:

- [es\\_lab/task\\_1/simulator/include/simulator.h](#)

## 6.2 CommServer Class Reference

Class for managing communication frontend and simulation backend.

```
#include <es_lab/task_1/simulator/include/server.h>
```

### Public Member Functions

- [CommServer](#) ([Simulator](#) &sim)  
*Constructor for [CommServer](#).*
- void [start\\_server](#) ()  
*Starts the communication server.*

### 6.2.1 Detailed Description

Class for managing communication frontend and simulation backend.

The [CommServer](#) class implements a communication server that handles HTTP requests to control and monitor the inverted pendulum simulation. It listens for incoming connections, processes HTTP requests, and sends corresponding responses.

### 6.2.2 Constructor & Destructor Documentation

#### 6.2.2.1 CommServer()

```
CommServer::CommServer (  
    Simulator & sim ) [inline]
```

Constructor for [CommServer](#).

Initializes the [CommServer](#) with the specified simulator object and listens for incoming connections on the specified IP address and port.

## Parameters

<i>sim</i>	Reference to the simulator object.
------------	------------------------------------

## 6.2.3 Member Function Documentation

### 6.2.3.1 start\_server()

```
void CommServer::start_server ( )
```

Starts the communication server.

This method starts the communication server in a separate thread.

The documentation for this class was generated from the following files:

- [es\\_lab/task\\_1/simulator/include/server.h](#)
- [es\\_lab/task\\_1/simulator/src/server.cpp](#)

## 6.3 Controller Class Reference

Interface for controllers used in the inverted pendulum simulation.

```
#include <es_lab/task_1/simulator/include/controller.h>
```

Inheritance diagram for Controller:

## 6.4 PIDController Class Reference

Implementation of a PID (Proportional-Integral-Derivative) controller.

```
#include <es_lab/task_1/simulator/include/controller.h>
```

Inheritance diagram for PIDController:

### Public Member Functions

- [PIDController](#) ()
- double [output](#) (double error)  
*Computes the control output based on the given error.*
- void [update\\_params](#) (double [kp](#), double [kd](#), double [ki](#))  
*Updates the controller parameters (gains).*
- void [setClamp](#) (double max, double min)  
*Sets the clamping limits for the control output.*

## Public Attributes

- double `kp` = 0.0  
*Default constructor.*
- double `kd` = 0.0
- double `ki` = 0.0

### 6.4.1 Detailed Description

Implementation of a PID (Proportional-Integral-Derivative) controller.

The `PIDController` class implements a PID controller for controlling the inverted pendulum system. It computes control signals based on proportional, integral, and derivative terms, and provides methods for updating controller parameters and setting clamping limits for the control output.

### 6.4.2 Constructor & Destructor Documentation

#### 6.4.2.1 `PIDController()`

```
PIDController::PIDController ( )
```

### 6.4.3 Member Function Documentation

#### 6.4.3.1 `output()`

```
double PIDController::output (
    double error ) [virtual]
```

Computes the control output based on the given error.

This method implements the PID control algorithm to compute the control output based on the error between the desired reference value and the current system state.

#### Parameters

<i>error</i>	The error (difference between reference value and current state).
--------------	---

#### Returns

The control output computed by the PID controller.

**Todo** Implement the PID controller output calculation

Implements [Controller](#).

### 6.4.3.2 setClamp()

```
void PIDController::setClamp (
    double max,
    double min ) [virtual]
```

Sets the clamping limits for the control output.

This method allows setting upper and lower limits for the control output to prevent excessive control action.

#### Parameters

<i>max</i>	The maximum allowed control signal.
<i>min</i>	The minimum allowed control signal.

**Todo** Implement setClamp for setting the output limits

Implements [Controller](#).

### 6.4.3.3 update\_params()

```
void PIDController::update_params (
    double kp,
    double kd,
    double ki ) [virtual]
```

Updates the controller parameters (gains).

This method allows for dynamic updating of the PID controller's gains (proportional, derivative, and integral gains).

#### Parameters

<i>kp</i>	The new proportional gain.
<i>kd</i>	The new derivative gain.
<i>ki</i>	The new integral gain.

**Todo** Implement the update\_params function for PID controller

Implements [Controller](#).

## 6.4.4 Member Data Documentation

#### 6.4.4.1 kd

```
double PIDController::kd = 0.0
```

#### 6.4.4.2 ki

```
double PIDController::ki = 0.0
```

#### 6.4.4.3 kp

```
double PIDController::kp = 0.0
```

Default constructor.

Initializes the PID controller with default proportional, derivative, and integral gains.

The documentation for this class was generated from the following files:

- es\_lab/task\_1/simulator/include/[controller.h](#)
- es\_lab/task\_1/simulator/src/[controller.cpp](#)

## 6.5 SimParams Struct Reference

Struct containing parameters for simulation.

```
#include <es_lab/task_1/simulator/include/simulator.h>
```

### Public Attributes

- double [simulation\\_time](#) = 1000  
*Duration of simulation in seconds.*
- double [delta\\_t](#) = 0.0001  
*Time step for simulation.*
- double [g](#) = 9.81  
*Acceleration due to gravity.*
- double [ref\\_angle](#)  
*Reference angle (0 is vertical, must be between -pi and pi)*
- int [delay](#) = 0  
*Delay in microseconds.*
- int [jitter](#) = 0  
*Jitter in microseconds.*

### 6.5.1 Detailed Description

Struct containing parameters for simulation.

### 6.5.2 Member Data Documentation

#### 6.5.2.1 delay

```
int SimParams::delay = 0
```

Delay in microseconds.

#### 6.5.2.2 delta\_t

```
double SimParams::delta_t = 0.0001
```

Time step for simulation.

#### 6.5.2.3 g

```
double SimParams::g = 9.81
```

Acceleration due to gravity.

#### 6.5.2.4 jitter

```
int SimParams::jitter = 0
```

Jitter in microseconds.

#### 6.5.2.5 ref\_angle

```
double SimParams::ref_angle
```

**Initial value:**

```
=  
    M_PI_4 /  
    8
```

Reference angle (0 is vertical, must be between -pi and pi)

### 6.5.2.6 simulation\_time

```
double SimParams::simulation_time = 1000
```

Duration of simulation in seconds.

The documentation for this struct was generated from the following file:

- [es\\_lab/task\\_1/simulator/include/simulator.h](#)

## 6.6 Simulator Class Reference

[Simulator](#) class for simulating the inverted pendulum.

```
#include <es_lab/task_1/simulator/include/simulator.h>
```

### Public Member Functions

- [Simulator](#) ()  
*Deleted default constructor.*
- [Simulator](#) (std::unique\_ptr< [Controller](#) > controller, const [SimParams](#) &params, const [Cart](#) &cart)  
*Constructs a [Simulator](#) object with the specified controller, simulation parameters, and cart parameters.*
- void [run\\_simulator](#) ()  
*Runs the simulator.*
- void [update\\_params](#) (double ref, int delay, int jitter)  
*Updates the simulation parameters. Function is called by the communication server to update the simulation parameters, once the client sends the new parameters.*
- void [reset\\_simulator](#) ()  
*Resets the simulator to its initial state. Function is called by the communication server to reset the simulator, once the client sends the reset command. This function resets the state variables of the simulator, allowing for a fresh start of the simulation.*

### Public Attributes

- std::unique\_ptr< [Controller](#) > [m\\_controller](#)  
*[Controller](#) object.*
- [SimParams](#) [m\\_params](#)  
*Simulation Parameters.*
- [Cart](#) [m\\_cart](#)  
*[Cart](#) object.*
- std::atomic< bool > [g\\_start](#) {false}  
*Flag to start the simulation.*
- std::atomic< bool > [g\\_reset](#) {false}  
*Flag to reset the simulation.*
- std::atomic< bool > [g\\_pause](#) {true}  
*Flag to pause the simulation.*
- std::condition\_variable [g\\_pause\\_cv](#)  
*Condition variable for pausing the simulation.*
- std::condition\_variable [g\\_start\\_cv](#)



- *Condition variable for starting the simulation.*
- std::mutex [g\\_start\\_mutex](#)
- std::mutex [g\\_pause\\_mutex](#)
- double [T](#) = 0
- *Current simulation time.*
- double [F](#) = 0
- *External force on the cart.*
- const int [buffer\\_size](#)
- *Size of the circular buffer for storing theta values.*
- std::array< double, 100 > [theta](#)
- *Circular buffer to store values of theta.*
- std::array< double, 2 > [theta\\_dot](#)
- *Last two angular velocities of the pendulum.*
- std::array< double, 2 > [theta\\_dot\\_dot](#)
- *Last two angular accelerations of the pendulum.*
- std::array< double, 2 > [x](#) {0, 0}
- *Last two positions of the cart.*
- std::array< double, 2 > [x\\_dot](#) {0, 0}
- *Last two velocities of the cart.*
- std::array< double, 2 > [x\\_dot\\_dot](#) {0, 0}
- *Last two accelerations of the cart.*
- const double [c\\_ml](#) = m\_cart.m \* m\_cart.len
- *Constant term.*
- const double [B](#) = m\_cart.M + m\_cart.m
- *Constant term.*
- const double [a](#)
- *Constant term.*
- double [A](#) = 0
- *State variable.*
- double [b](#) = 0
- *State variable.*
- double [C](#) = 0
- *State variable.*
- double [c](#) = 0
- *State variable.*
- double [E](#) = 0
- *Total energy of the system.*
- double [error](#) = 0
- *Difference between reference angle and current angle.*
- int [i](#) = 0
- *Index in circular buffer for theta values.*

### 6.6.1 Detailed Description

[Simulator](#) class for simulating the inverted pendulum.

### 6.6.2 Constructor & Destructor Documentation

### 6.6.2.1 Simulator() [1/2]

```
Simulator::Simulator ( ) [inline]
```

Deleted default constructor.

### 6.6.2.2 Simulator() [2/2]

```
Simulator::Simulator (
    std::unique_ptr< Controller > controller,
    const SimParams & params,
    const Cart & cart ) [inline]
```

Constructs a [Simulator](#) object with the specified controller, simulation parameters, and cart parameters.

#### Parameters

<i>controller</i>	Reference to the controller
<i>params</i>	Reference to the simulation parameters
<i>cart</i>	Reference to the cart parameters

## 6.6.3 Member Function Documentation

### 6.6.3.1 reset\_simulator()

```
void Simulator::reset_simulator ( )
```

Resets the simulator to its initial state. Function is called by the communication server to reset the simulator, once the client sends the reset command. This function resets the state variables of the simulator, allowing for a fresh start of the simulation.

### 6.6.3.2 run\_simulator()

```
void Simulator::run_simulator ( )
```

Runs the simulator.

This function starts the simulation loop, updating the state of the system at each time step based on the controller output and simulation parameters.

**Todo** Implement delay and jitter by changing delay index based on [SimParams.delay](#) and [SimParams.jitter](#)

**Todo** Handle case when delay index is negative, wrap around to end of circular buffer

**Todo**

**Todo** Make sure delay index is within bounds of buffer size

### 6.6.3.3 update\_params()

```
void Simulator::update_params (
    double ref = 0,
    int delay = 0,
    int jitter = 0 )
```

Updates the simulation parameters. Function is called by the communication server to update the simulation parameters, once the client sends the new parameters.

#### Parameters

<i>ref</i>	Reference angle for the inverted pendulum.
<i>delay</i>	Delay in microseconds for synchronization with the communication server.
<i>jitter</i>	Jitter in microseconds for synchronization with the communication server.

**Todo** Implement update\_params function to update simulation parameters

## 6.6.4 Member Data Documentation

### 6.6.4.1 a

```
const double Simulator::a
```

#### Initial value:

```
=
    m_cart.I + m_cart.m * std::pow(m_cart.len, 2)
```

Constant term.

### 6.6.4.2 A

```
double Simulator::A = 0
```

State variable.

### 6.6.4.3 B

```
const double Simulator::B = m_cart.M + m_cart.m
```

Constant term.

#### 6.6.4.4 **b**

```
double Simulator::b = 0
```

State variable.

#### 6.6.4.5 **buffer\_size**

```
const int Simulator::buffer_size
```

**Initial value:**

```
=  
    100
```

Size of the circular buffer for storing theta values.

#### 6.6.4.6 **C**

```
double Simulator::C = 0
```

State variable.

#### 6.6.4.7 **c**

```
double Simulator::c = 0
```

State variable.

#### 6.6.4.8 **c\_ml**

```
const double Simulator::c_ml = m_cart.m * m_cart.len
```

Constant term.

#### 6.6.4.9 **E**

```
double Simulator::E = 0
```

Total energy of the system.

#### 6.6.4.10 error

```
double Simulator::error = 0
```

Difference between reference angle and current angle.

#### 6.6.4.11 F

```
double Simulator::F = 0
```

External force on the cart.

#### 6.6.4.12 g\_pause

```
std::atomic<bool> Simulator::g_pause {true}
```

Flag to pause the simulation.

#### 6.6.4.13 g\_pause\_cv

```
std::condition_variable Simulator::g_pause_cv
```

Condition variable for pausing the simulation.

#### 6.6.4.14 g\_pause\_mutex

```
std::mutex Simulator::g_pause_mutex
```

Mutex for Synchronization of simulation pause between simulator and comm server

#### 6.6.4.15 g\_reset

```
std::atomic<bool> Simulator::g_reset {false}
```

Flag to reset the simulation.

#### 6.6.4.16 g\_start

```
std::atomic<bool> Simulator::g_start {false}
```

Flag to start the simulation.

#### 6.6.4.17 g\_start\_cv

```
std::condition_variable Simulator::g_start_cv
```

Condition variable for starting the simulation.

#### 6.6.4.18 g\_start\_mutex

```
std::mutex Simulator::g_start_mutex
```

Mutex for Synchronization of simulation start between simulator and comm server

#### 6.6.4.19 i

```
int Simulator::i = 0
```

Index in circular buffer for theta values.

#### 6.6.4.20 m\_cart

```
Cart Simulator::m_cart
```

[Cart](#) object.

#### 6.6.4.21 m\_controller

```
std::unique_ptr<Controller> Simulator::m_controller
```

[Controller](#) object.

#### 6.6.4.22 m\_params

```
SimParams Simulator::m_params
```

Simulation Parameters.

#### 6.6.4.23 T

```
double Simulator::T = 0
```

Current simulation time.

#### 6.6.4.24 theta

```
std::array<double, 100> Simulator::theta
```

Circular buffer to store values of theta.

#### 6.6.4.25 theta\_dot

```
std::array<double, 2> Simulator::theta_dot
```

**Initial value:**

```
{  
    0, 0}
```

Last two angular velocities of the pendulum.

#### 6.6.4.26 theta\_dot\_dot

```
std::array<double, 2> Simulator::theta_dot_dot
```

**Initial value:**

```
{  
    0, 0}
```

Last two angular accelerations of the pendulum.

**6.6.4.27 x**

```
std::array<double, 2> Simulator::x {0, 0}
```

Last two positions of the cart.

**6.6.4.28 x\_dot**

```
std::array<double, 2> Simulator::x_dot {0, 0}
```

Last two velocities of the cart.

**6.6.4.29 x\_dot\_dot**

```
std::array<double, 2> Simulator::x_dot_dot {0, 0}
```

Last two accelerations of the cart.

The documentation for this class was generated from the following files:

- es\_lab/task\_1/simulator/include/[simulator.h](#)
- es\_lab/task\_1/simulator/src/[simulator.cpp](#)



# Chapter 7

## File Documentation

### 7.1 es\_lab/task\_1/simulator/include/controller.h File Reference

Header file for the [Controller](#) interface and [PIDController](#) class.

This graph shows which files directly or indirectly include this file:

#### Classes

- class [Controller](#)  
*Interface for controllers used in the inverted pendulum simulation.*
- class [PIDController](#)  
*Implementation of a PID (Proportional-Integral-Derivative) controller.*

#### 7.1.1 Detailed Description

Header file for the [Controller](#) interface and [PIDController](#) class.

This file declares the [Controller](#) interface, which defines the interface for controllers used in the inverted pendulum simulation. It also declares the [PIDController](#) class, which implements a PID (Proportional-Integral-Derivative) controller for controlling the pendulum system.

#### Author

Utkarsh Raj

#### Date

10-April-2024

## 7.2 controller.h

[Go to the documentation of this file.](#)

```

1
15 #pragma once
16
24 class Controller {
25 public:
36     virtual double output(double error) = 0;
47     virtual void update_params(double kp, double kd, double ki) = 0;
48
58     virtual void setClamp(double max, double min) = 0;
59 };
60
69 class PIDController : public Controller {
70     // @todo Add private members for PIDController class
71 public:
78     double kp = 0.0;
79     double kd = 0.0;
80     double ki = 0.0;
81     PIDController();
82
94     double output(double error);
105     void update_params(double kp, double kd, double ki);
115     void setClamp(double max, double min);
116 };

```

## 7.3 es\_lab/task\_1/simulator/include/server.h File Reference

Header file for the [CommServer](#) class.

```

#include "boost/asio/ip/tcp.hpp"
#include <boost/beast/core.hpp>
#include <boost/beast/http.hpp>
#include <boost/beast/version.hpp>
#include <boost/beast/websocket.hpp>
#include <boost/config.hpp>
#include <iostream>
#include <json.hpp>
#include "controller.h"
#include "simulator.h"
#include <mutex>
#include <thread>

```

Include dependency graph for server.h: This graph shows which files directly or indirectly include this file:

### Classes

- class [CommServer](#)

*Class for managing communication frontend and simulation backend.*

### Typedefs

- using [json](#) = nlohmann::json
- using [tcp](#) = boost::asio::ip::tcp

### 7.3.1 Detailed Description

Header file for the `CommServer` class.

This file declares the `CommServer` class, which implements a communication server for interacting with the inverted pendulum simulation. It utilizes Boost.Asio for asynchronous I/O operations and Boost.Beast for handling HTTP requests and responses. The `CommServer` class facilitates communication with the simulation, allowing control and monitoring of simulation parameters over network.

#### Author

Utkarsh Raj

#### Date

10-April-2024

### 7.3.2 Typedef Documentation

#### 7.3.2.1 json

```
using json = nlohmann::json
```

#### 7.3.2.2 tcp

```
using tcp = boost::asio::ip::tcp
```

## 7.4 server.h

[Go to the documentation of this file.](#)

```
1
16 #include "boost/asio/ip/tcp.hpp"
17 #include <boost/beast/core.hpp>
18 #include <boost/beast/http.hpp>
19 #include <boost/beast/version.hpp>
20 #include <boost/beast/websocket.hpp>
21 #include <boost/config.hpp>
22 #include <iostream>
23 #include <json.hpp>
24
25 #include "controller.h"
26 #include "simulator.h"
27 #include <mutex>
28 #include <thread>
29
30 using json = nlohmann::json;
31
32 namespace beast = boost::beast;    // from <boost/beast.hpp>
33 namespace http = beast::http;      // from <boost/beast/http.hpp>
34 namespace net = boost::asio;       // from <boost/asio.hpp>
35 using tcp = boost::asio::ip::tcp;  // from <boost/asio/ip/tcp.hpp>
36
45 class CommServer {
```

```

46  Simulator &sim;
47
48  net::io_context ioc{1};
49
50  net::ip::address address{
51      net::ip::make_address("0.0.0.0")};
52  unsigned short port{8080};
53  tcp::acceptor
54      acceptor;
55 public:
56  CommServer(Simulator &sim) : sim(sim), acceptor(ioc, {address, port}) {}
57  void start_server();
58
59 private:
60  void run_server();
61  void handle_request(tcp::socket &socket);
62 };

```

## 7.5 es\_lab/task\_1/simulator/include/simulator.h File Reference

Header file for [Simulator](#) class, [SimParams](#) and [Cart](#).

```

#include "controller.h"
#include <array>
#include <atomic>
#include <cmath>
#include <condition_variable>
#include <memory>
#include <mutex>

```

Include dependency graph for simulator.h: This graph shows which files directly or indirectly include this file:

### Classes

- struct [SimParams](#)  
*Struct containing parameters for simulation.*
- struct [Cart](#)  
*Struct containing parameters for the cart.*
- class [Simulator](#)  
*Simulator class for simulating the inverted pendulum.*

### 7.5.1 Detailed Description

Header file for [Simulator](#) class, [SimParams](#) and [Cart](#).

This file contains declarations for the simulation parameters struct, [Cart](#) Struct and the [Simulator](#) class, which are used for simulating the behavior of an inverted pendulum system. It also includes declarations for related data structures and synchronization primitives used in the simulation.

#### Author

Utkarsh Raj

#### Date

10-April-2024

## 7.6 simulator.h

[Go to the documentation of this file.](#)

```

1
14 #pragma once
15
16 #include "controller.h"
17 #include <array>
18 #include <atomic>
19 #include <cmath>
20 #include <condition_variable>
21 #include <memory>
22 #include <mutex>
23
24 struct SimParams {
25     double simulation_time = 1000;
26     double delta_t = 0.0001;
27     double g = 9.81;
28     double ref_angle =
29         M_PI_4 /
30         8;
31     int delay = 0;
32     int jitter = 0;
33 };
34
35 struct Cart {
36     double M = 5;
37     double m = 0.5;
38     double len = 1;
39     double I = m * len * len;
40 };
41
42 class Simulator {
43 public:
44     std::unique_ptr<Controller> m_controller;
45     SimParams m_params;
46     Cart m_cart;
47
48     // Synchronization variables between simulator and comm server
49     std::atomic<bool> g_start{false};
50     std::atomic<bool> g_reset{false};
51     std::atomic<bool> g_pause{true};
52     std::condition_variable
53         g_pause_cv;
54     std::condition_variable
55         g_start_cv;
56     std::mutex g_start_mutex;
57     std::mutex g_pause_mutex;
58
59     // run time variables
60     double T = 0;
61     double F = 0;
62
63     // State of the pendulum
64     const int buffer_size =
65         100;
66     std::array<double, 100> theta;
67     std::array<double, 2> theta_dot{
68         0, 0};
69     std::array<double, 2> theta_dot_dot{
70         0, 0};
71
72     // State of the cart
73     std::array<double, 2> x{0, 0};
74     std::array<double, 2> x_dot{0, 0};
75     std::array<double, 2> x_dot_dot{0, 0};
76
77     // Generic solution to system of two equations
78     const double c_ml = m_cart.m * m_cart.len;
79     const double B = m_cart.M + m_cart.m;
80     const double a =
81         m_cart.I + m_cart.m * std::pow(m_cart.len, 2);
82
83     // updates every time step
84     double A = 0;
85     double b = 0;
86     double C = 0;
87     double c = 0;
88     double E = 0;
89     double error = 0;
90     int i = 0;
91
92     Simulator() : m_controller(std::make_unique<PIDController>()) {
93         theta.fill(0);

```

```

109     theta.at(0) = M_PI_4 / 8;
110 };
111
120 Simulator(std::unique_ptr<Controller> controller, const SimParams &params,
121           const Cart &cart)
122     : m_controller(std::move(controller)), m_params(params), m_cart(cart) {}
129 void run_simulator();
130
141 void update_params(double ref, int delay, int jitter);
142
149 void reset_simulator();
150 };

```

## 7.7 es\_lab/task\_1/simulator/README.md File Reference

## 7.8 es\_lab/task\_1/simulator/src/controller.cpp File Reference

Implementation file for the [PIDController](#) class.

```

#include "controller.h"
#include <iostream>

```

Include dependency graph for controller.cpp:

### 7.8.1 Detailed Description

Implementation file for the [PIDController](#) class.

This file contains the implementation of the [PIDController](#) class, which implements a PID (Proportional-Integral-Derivative) controller for controlling the inverted pendulum system.

#### Author

[Your Name]

#### Date

[Date]

## 7.9 es\_lab/task\_1/simulator/src/main.cpp File Reference

Main entry point for the inverted pendulum simulation.

```

#include "controller.h"
#include "server.h"
#include "simulator.h"
#include <thread>

```

Include dependency graph for main.cpp:

### Functions

- int [main](#) ()

*Main function to start the simulation and communication server.*

### 7.9.1 Detailed Description

Main entry point for the inverted pendulum simulation.

This file contains the main function, which serves as the entry point for the simulation of an inverted pendulum system. It initializes simulator object, and communication server, and starts the simulation and communication server threads.

#### Author

Utkarsh Raj

#### Date

10-April-2024

### 7.9.2 Function Documentation

#### 7.9.2.1 main()

```
int main ( )
```

Main function to start the simulation and communication server.

This function initializes the simulator object and communication server. It then starts the simulation and communication server threads, waits for them to finish.

#### Returns

0 on successful completion.

< [Simulator](#) object

< Communication server object with simulator object

< Start the simulation thread

< Start the communication server thread

< Wait for the threads to finish

## 7.10 es\_lab/task\_1/simulator/src/server.cpp File Reference

Implementation file for the [CommServer](#) class.

```
#include "server.h"
```

Include dependency graph for server.cpp:

### 7.10.1 Detailed Description

Implementation file for the [CommServer](#) class.

This file contains the implementation of the methods declared in the [server.h](#) header file. It provides the functionality for handling incoming HTTP requests and managing communication with the inverted pendulum simulation.

**Author**

Utkarsh Raj

**Date**

10-April-2024

## 7.11 es\_lab/task\_1/simulator/src/simulator.cpp File Reference

Implementation file for the [Simulator](#) class.

```
#include "simulator.h"
#include "controller.h"
#include <chrono>
#include <cmath>
#include <experimental/random>
#include <iostream>
#include <mutex>
#include <thread>
Include dependency graph for simulator.cpp:
```

### 7.11.1 Detailed Description

Implementation file for the [Simulator](#) class.

This file contains the implementation of the [Simulator](#) class, which simulates the behavior of an inverted pendulum system. It includes the definition of member functions for running the simulation, updating simulation parameters, and resetting the simulator to its initial state.

**Author**

Utkarsh Raj

**Date**

10-April-2024



# Index

- A
  - Simulator, [21](#)
- a
  - Simulator, [21](#)
- B
  - Simulator, [21](#)
- b
  - Simulator, [21](#)
- buffer\_size
  - Simulator, [22](#)
- C
  - Simulator, [22](#)
- c
  - Simulator, [22](#)
- c\_ml
  - Simulator, [22](#)
- Cart, [11](#)
  - l, [11](#)
  - len, [11](#)
  - M, [12](#)
  - m, [12](#)
- CommServer, [12](#)
  - CommServer, [12](#)
  - start\_server, [13](#)
- Controller, [13](#)
- delay
  - SimParams, [17](#)
- delta\_t
  - SimParams, [17](#)
- E
  - Simulator, [22](#)
- error
  - Simulator, [22](#)
- es\_lab/task\_1/simulator/include/controller.h, [27](#), [28](#)
- es\_lab/task\_1/simulator/include/server.h, [28](#), [29](#)
- es\_lab/task\_1/simulator/include/simulator.h, [30](#), [31](#)
- es\_lab/task\_1/simulator/README.md, [32](#)
- es\_lab/task\_1/simulator/src/controller.cpp, [32](#)
- es\_lab/task\_1/simulator/src/main.cpp, [32](#)
- es\_lab/task\_1/simulator/src/server.cpp, [33](#)
- es\_lab/task\_1/simulator/src/simulator.cpp, [34](#)
- F
  - Simulator, [23](#)
- g
  - SimParams, [17](#)
- g\_pause
  - Simulator, [23](#)
- g\_pause\_cv
  - Simulator, [23](#)
- g\_pause\_mutex
  - Simulator, [23](#)
- g\_reset
  - Simulator, [23](#)
- g\_start
  - Simulator, [23](#)
- g\_start\_cv
  - Simulator, [24](#)
- g\_start\_mutex
  - Simulator, [24](#)
- I
  - Cart, [11](#)
- i
  - Simulator, [24](#)
- jitter
  - SimParams, [17](#)
- json
  - server.h, [29](#)
- kd
  - PIDController, [15](#)
- ki
  - PIDController, [16](#)
- kp
  - PIDController, [16](#)
- len
  - Cart, [11](#)
- M
  - Cart, [12](#)
- m
  - Cart, [12](#)
- m\_cart
  - Simulator, [24](#)
- m\_controller
  - Simulator, [24](#)
- m\_params
  - Simulator, [24](#)
- main
  - main.cpp, [33](#)
- main.cpp
  - main, [33](#)
- output

- PIDController, 14
- PIDController, 13
  - kd, 15
  - ki, 16
  - kp, 16
  - output, 14
  - PIDController, 14
  - setClamp, 15
  - update\_params, 15
- ref\_angle
  - SimParams, 17
- reset\_simulator
  - Simulator, 20
- run\_simulator
  - Simulator, 20
- server.h
  - json, 29
  - tcp, 29
- setClamp
  - PIDController, 15
- SimParams, 16
  - delay, 17
  - delta\_t, 17
  - g, 17
  - jitter, 17
  - ref\_angle, 17
  - simulation\_time, 17
- simulation\_time
  - SimParams, 17
- Simulator, 18
  - A, 21
  - a, 21
  - B, 21
  - b, 21
  - buffer\_size, 22
  - C, 22
  - c, 22
  - c\_ml, 22
  - E, 22
  - error, 22
  - F, 23
  - g\_pause, 23
  - g\_pause\_cv, 23
  - g\_pause\_mutex, 23
  - g\_reset, 23
  - g\_start, 23
  - g\_start\_cv, 24
  - g\_start\_mutex, 24
  - i, 24
  - m\_cart, 24
  - m\_controller, 24
  - m\_params, 24
  - reset\_simulator, 20
  - run\_simulator, 20
  - Simulator, 19, 20
  - T, 25
  - theta, 25
  - theta\_dot, 25
  - theta\_dot\_dot, 25
  - update\_params, 20
  - x, 25
  - x\_dot, 26
  - x\_dot\_dot, 26
- start\_server
  - CommServer, 13
- T
  - Simulator, 25
- tcp
  - server.h, 29
- theta
  - Simulator, 25
- theta\_dot
  - Simulator, 25
- theta\_dot\_dot
  - Simulator, 25
- update\_params
  - PIDController, 15
  - Simulator, 20
- x
  - Simulator, 25
- x\_dot
  - Simulator, 26
- x\_dot\_dot
  - Simulator, 26