Embedded Systems Lab Task 1

PID Control of Inverted Pendulum

Chapter 1

Getting Started

1.1 Table of Contents

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 (kp, kd, ki).
- 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, relase 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. Go to `eslab1.pages.dev`
4. Use the web interface to control and monitor the simulation parameters. You can start, stop, and adjust the simulation settings as needed.
5. Interact with the simulator via the web interface to observe the behavior of the inverted pendulum system under different control conditions.

Alternatively, you can run the simulator with the frontend server locally to control and monitor the simulation via a web interface:

1. Download the frontend code from `github`
2. Follow the instructions in the frontend README to start the frontend server.

- Install npm and serve package and run the serve command in the dist directory

```
npm install -g serve
serve
```

- On Windows, runinng *serve* in powsershell mau require the following command to be run first

```
Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Scope CurrentUser
```

3. Open a web browser and navigate to the frontend server address (default: http://localhost:3000).

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

1.7 Online Documentation

Online version of this Code documentation can be found at eslab1doc.pages.dev

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:

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

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
  Pendum 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
```

Pendum 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

sim	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

error	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

max	The maximum allowed control signal.
min	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

kp	The new proportional gain.
kd	The new derivative gain.
ki	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

controller	Reference to the controller
params	Reference to the simulation parameters
cart	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

ref	Reference angle for the inverted pendulum.
delay	Delay in microseconds for synchronization with the communication server.
jitter	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.

```
#pragma once

class Controller {
public:
  virtual double output(double error) = 0;
  virtual void update_params(double kp, double kd, double ki) = 0;

  virtual void setClamp(double max, double min) = 0;
};

class PIDController : public Controller {
  //@todo Add private members for PIDController class
public:
  double kp = 0.0;
  double kd = 0.0;
  double ki = 0.0;
  PIDController();

  double output(double error);
   void update_params(double kp, double kd, double ki);
   void setClamp(double max, double min);
};
```

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.

```

#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>

using json = nlohmann::json;

namespace beast = boost::beast;   // from <boost/beast.hpp>
namespace http = beast::http;     // from <boost/beast/http.hpp>
namespace net = boost::asio;      // from <boost/asio.hpp>
using tcp = boost::asio::ip::tcp; // from <boost/asio/ip/tcp.hpp>

class CommServer {
  Simulator &sim;
```

```

  net::io_context ioc{1};

  net::ip::address address{
      net::ip::make_address("0.0.0.0")};
  unsigned short port{8080};
  tcp::acceptor
      acceptor;
public:
  CommServer(Simulator &sim) : sim(sim), acceptor(ioc, {address, port}) {}
  void start_server();

private:
  void run_server();
  void handle_request(tcp::socket &socket);
};
```

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.

```
#pragma once

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

struct SimParams {
  double simulation_time = 1000;
  double delta_t = 0.0001;
  double g = 9.81;
  double ref_angle =
      M_PI_4 /
      8;
  int delay = 0;
  int jitter = 0;
};

struct Cart {
  double M = 5;
  double m = 0.5;
  double len = 1;
  double I = m * len * len;
};

class Simulator {

public:
  std::unique_ptr<Controller> m_controller;
  SimParams m_params;
  Cart m_cart;

  // Synchronization variables between simulator and comm server
  std::atomic<bool> g_start{false};
  std::atomic<bool> g_reset{false};
  std::atomic<bool> g_pause{true};
  std::condition_variable
      g_pause_cv;
  std::condition_variable
      g_start_cv;
  std::mutex g_start_mutex;
  std::mutex g_pause_mutex;

  // run time variables
  double T = 0;
  double F = 0;

  // State of the pendulum
  const int buffer_size =
      100;
  std::array<double, 100> theta;
  std::array<double, 2> theta_dot{
      0, 0};
  std::array<double, 2> theta_dot_dot{
      0, 0};

  // State of the cart
  std::array<double, 2> x{0, 0};
  std::array<double, 2> x_dot{0, 0};
  std::array<double, 2> x_dot_dot{0, 0};

  // Generic solution to system of two equations
  const double c_ml = m_cart.m * m_cart.len;
  const double B = m_cart.M + m_cart.m;
  const double a =
      m_cart.I + m_cart.m * std::pow(m_cart.len, 2);

  // updates every time step
  double A = 0;
  double b = 0;
  double C = 0;
  double c = 0;
  double E = 0;
  double error = 0;
  int i = 0;

  Simulator() : m_controller(std::make_unique<PIDController>()) {
    theta.fill(0);
```

```
      theta.at(0) = M_PI_4 / 8;
    };

    Simulator(std::unique_ptr<Controller> controller, const SimParams &params,
              const Cart &cart)
        : m_controller(std::move(controller)), m_params(params), m_cart(cart) {}
    void run_simulator();

    void update_params(double ref, int delay, int jitter);

    void reset_simulator();
};
```

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