Control of Inverted Pendulum on Cart using Julia

Melaku Yemaneberhan MESIHU

January 21, 2025

Contents

1 Introduction

This work presents an **educational toolbox** in the **Julia** language for demonstrating and experimenting with the classic "inverted pendulum on a cart" (or "cart-pole") control problem. Julia has rapidly emerged as a high-level language that combines the productivity of scripting languages with the performance of lower-level languages, making it an excellent choice for scientific computing and interactive applications.

Julia is a high-level programming language that offers near-native code performance without sacrificing ease of use. It is well-suited for data-intensive tasks with built-in support for arrays, linear algebra, and parallelization. By leveraging just-in-time compilation, it enables scientists and engineers to write expressive code that can run at remarkable speeds. Multiple dispatch, a key feature, makes it easy to write highly generic and efficient functions for diverse data types. The growing ecosystem provides robust tools for optimization, visualization, and machine learning, making it an ideal choice for many scientific and technical applications.

Why Julia, Makie, GTK, and Pluto?

- **Julia**: Offers just-in-time compilation (via LLVM), a rich ecosystem of packages (including `DifferentialEquations.jl` and `ControlSystems.jl`), and a syntax well suited for mathematical modeling.
- **Makie.jl**: Provides high-performance, interactive plotting and real-time animations. We use `GLMakie` for rendering the cart-pole simulation and updating the scene as states evolve.
- **GTK**: A toolkit that can be used with Julia to build graphical user interfaces. Although not strictly required if using Makie's built-in displays, GTK is an option for creating more customized windows, dialogs, or widgets.
- **Pluto.jl**: A reactive notebook environment for Julia, enabling live interactivity. Parameters such as controller gains, pendulum mass, or simulation time can be manipulated with sliders or text fields, and the simulation re-runs automatically. This fosters an engaging, hands-on learning experience.

In my implementation, I combine Makie's powerful visualization capabilities with Julia's robust ODE solvers and control libraries. The end result is a stand-alone simulation GUI that can also be integrated into a Pluto notebook for interactive demos. Users can experiment with different controller types (PID or LQR), adjust system parameters on the fly, and instantly see the cart-pole's response through real-time plots and animations.

2 Objectives

The primary goal is to create a **versatile and interactive educational toolbox in Julia** for the cart-pole system. Specifically:

1. Develop a **user-friendly GUI** that allows students or researchers to change system parameters (cart mass, pendulum length, etc.) and controller gains intuitively.
2. Showcase **two fundamental control strategies** (PID and LQR) for balancing and positioning the inverted pendulum.
3. Provide **real-time visualization** and interactive plots to enhance understanding of system dynamics and control performance.
4. Facilitate **hands-on experimentation** a custom GUI, ensuring a smooth workflow for educational purposes.

3 Mathematical Modeling

3.1 System Variables

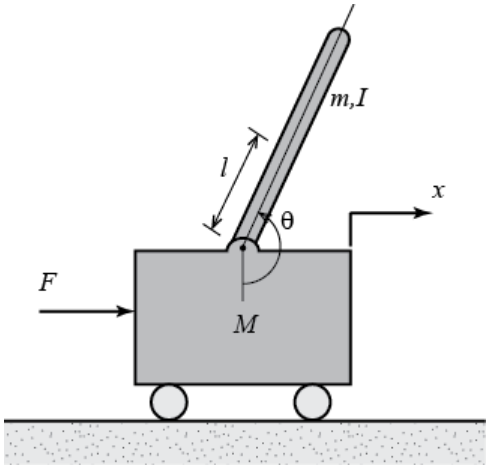

Figure 1: Inverted Pendulum on a Cart [4]

To describe the cart-pole system:

- x: the horizontal position of the cart (meters). Positive x is typically to the right.
- $\dot{x}$: the horizontal velocity of the cart (m/s).
- θ: the angle of the pendulum (radians). We define $\theta = \pi$ as the upright (inverted) position.
- $\dot{\theta}$: the angular velocity of the pendulum (rad/s).
- M: the mass of the cart (kg).
- m: the mass of the pendulum bob (kg).

- ℓ: the length of the pendulum (m).
- b: a linear damping coefficient (kg/s) modeling friction on the cart's motion.
- g: the gravitational acceleration (m/s^2), assumed constant at $9.81\,\mathrm{m/s^2}$.
- F: the horizontal control force (newtons) applied to the cart.

Hence, the system state can be written as

$$\mathbf{x} = \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix}.$$

3.2 Detailed Derivation of the Equations of Motion

This section shows how we arrive at the well-known cart-pole equations of motion. We will use the *Lagrangian approach* and then incorporate the external (non-conservative) force F and friction $b\,\dot{x}$.

1) Kinetic and Potential Energy

Let us place the cart on a frictional track, moving horizontally with coordinate $x(t)$. The pivot of the pendulum is attached to the cart; the pendulum mass m has an angle θ as measured from the downward vertical or upward vertical. In our case, we define $\theta = \pi$ to be *upright*, so if $\theta = 0$ is downward, the geometry is consistent with:

$$\text{(Pendulum bob coordinates)} \quad \begin{cases} X_{\text{bob}} = x + \ell \sin(\theta), \\ Y_{\text{bob}} = -\,\ell \cos(\theta), \end{cases}$$

where we might define the origin so that $Y = 0$ is at the pivot when $\theta = 0$. (The exact sign convention depends on your diagram; you only need internal consistency.)

Kinetic Energy (T).

- The cart of mass M has horizontal velocity $\dot{x}$.
- The pendulum mass m has velocity components

$$\dot{X}_{\text{bob}} = \dot{x} + \ell \cos(\theta)\,\dot{\theta}, \quad \dot{Y}_{\text{bob}} = \ell \sin(\theta)\,\dot{\theta}.$$

  Hence, the pendulum bob's speed is $\sqrt{\dot{X}_{\text{bob}}^2 + \dot{Y}_{\text{bob}}^2}$.

  Therefore,

$$T = \frac{1}{2} M\,\dot{x}^2 + \frac{1}{2} m\big(\dot{X}_{\text{bob}}^2 + \dot{Y}_{\text{bob}}^2\big).$$

Substitute the velocities:

$$\dot{X}_{\text{bob}}^2 = \big(\dot{x} + \ell \cos(\theta)\,\dot{\theta}\big)^2, \quad \dot{Y}_{\text{bob}}^2 = \big(\ell \sin(\theta)\,\dot{\theta}\big)^2.$$

Expanding,

$$T = \frac{1}{2}\, M\, \dot{x}^2 + \frac{1}{2}\, m\big[\dot{x}^2 + 2\,\dot{x}\,\ell\cos(\theta)\,\dot{\theta} + \ell^2\cos^2(\theta)\,\dot{\theta}^2 + \ell^2\sin^2(\theta)\,\dot{\theta}^2\big].$$

Combine like terms, noting $\cos^2(\theta) + \sin^2(\theta) = 1$:

$$T = \frac{1}{2}\,(M+m)\,\dot{x}^2 + m\,\ell\,\dot{x}\,\cos(\theta)\,\dot{\theta} + \tfrac{1}{2}\,m\,\ell^2\,\dot{\theta}^2.$$

Potential Energy (V). Taking gravitational potential as zero at $Y = 0$ for the pivot, the pendulum bob is at vertical coordinate $Y_{\text{bob}} = -\,\ell\cos(\theta)$. Thus:

$$V = m\,g\,(\ell - \ell\cos(\theta)) = m\,g\,\ell\,\big[1 - \cos(\theta)\big].$$

Often, one may ignore the constant $m\,g\,\ell$ (since it only shifts the total potential by a constant), but we keep it for clarity:

$$V = m\,g\,\ell\big[\,1 - \cos(\theta)\big].$$

2) Lagrangian and Generalized Forces

The *Lagrangian* is $L = T - V$. Hence,

$$L = \frac{1}{2}\,(M+m)\,\dot{x}^2 + m\,\ell\,\dot{x}\,\cos(\theta)\,\dot{\theta} + \tfrac{1}{2}\,m\,\ell^2\,\dot{\theta}^2 \;-\; m\,g\,\ell\,\big[1 - \cos(\theta)\big].$$

We have two generalized coordinates: x and θ. The friction force on the cart is $-\,b\,\dot{x}$, and the control force is F in the positive x-direction. Neither friction nor the applied force contributes directly to θ-torques, so the only generalized (non-conservative) force corresponding to x is

$$Q_x = F - b\,\dot{x}.$$

There is no generalized non-conservative force for θ, so $Q_\theta = 0$.

Euler-Lagrange Equations. The Euler-Lagrange equation for coordinate q is

$$\frac{d}{dt}\big(\tfrac{\partial L}{\partial \dot{q}}\big) \;-\; \tfrac{\partial L}{\partial q} = Q_q,$$

where Q_q is the generalized force associated with q. We apply this to $q = x$ and $q = \theta$.

3) Equation for x

$$\frac{d}{dt}\Big(\frac{\partial L}{\partial \dot{x}}\Big) - \frac{\partial L}{\partial x} = Q_x = F - b\,\dot{x}.$$

But

$$\frac{\partial L}{\partial \dot{x}} = (M+m)\,\dot{x} + m\,\ell\,\cos(\theta)\,\dot{\theta}, \quad \frac{\partial L}{\partial x} = 0$$

(since L does not explicitly depend on x, only on $\dot{x}$).

Hence

$$\frac{d}{dt}\Big[(M+m)\,\dot{x} + m\,\ell\,\cos(\theta)\,\dot{\theta}\Big] = F - b\,\dot{x}.$$

So

$$(M+m)\,\ddot{x} + m\,\ell\,\big[-\,\sin(\theta)\,\dot{\theta}^2 + \cos(\theta)\,\ddot{\theta}\big] = F - b\,\dot{x}.$$

Rewriting slightly,

$$(M+m)\,\ddot{x} - m\,\ell\,\cos(\theta)\,\ddot{\theta} + m\,\ell\,\sin(\theta)\,\dot{\theta}^2 + b\,\dot{x} = F.$$

4) Equation for θ

$$\frac{d}{dt}\Big(\frac{\partial L}{\partial \dot{\theta}}\Big) - \frac{\partial L}{\partial \theta} = Q_\theta = 0.$$

Compute:

$$\frac{\partial L}{\partial \dot{\theta}} = m\,\ell\,\dot{x}\,\cos(\theta) + m\,\ell^2\,\dot{\theta},$$

$$\frac{\partial L}{\partial \theta} = m\,\ell\,\big[-\dot{x}\,\sin(\theta)\,\dot{\theta}\big] + m\,\ell^2\,\dot{\theta}\,0 - m\,g\,\ell\,\sin(\theta).$$

Hence,

$$\frac{d}{dt}\Big[m\,\ell\,\dot{x}\,\cos(\theta) + m\,\ell^2\,\dot{\theta}\Big] - \Big[-m\,\ell\,\dot{x}\,\sin(\theta)\,\dot{\theta} - m\,g\,\ell\,\sin(\theta)\Big] = 0.$$

Simplify. The derivative w.r.t. t is:

$$m\,\ell\,\big[\ddot{x}\,\cos(\theta) - \dot{x}\,\sin(\theta)\,\dot{\theta}\big] + m\,\ell^2\,\ddot{\theta}.$$

So the expression is

$$m\,\ell\,\ddot{x}\,\cos(\theta) - m\,\ell\,\dot{x}\,\sin(\theta)\,\dot{\theta} + m\,\ell^2\,\ddot{\theta} + m\,\ell\,\dot{x}\,\sin(\theta)\,\dot{\theta} + m\,g\,\ell\,\sin(\theta) = 0.$$

Notice that $\pm m\,\ell\,\dot{x}\,\sin(\theta)\,\dot{\theta}$ terms cancel out, leaving

$$m\,\ell\,\ddot{x}\,\cos(\theta) + m\,\ell^2\,\ddot{\theta} + m\,g\,\ell\,\sin(\theta) = 0.$$

Or

$$m\,\ell^2\,\ddot{\theta} + m\,\ell\,\ddot{x}\,\cos(\theta) + m\,g\,\ell\,\sin(\theta) = 0.$$

We can solve for $\ddot{\theta}$ once $\ddot{x}$ is known (or we can couple the two equations). The standard references solve them simultaneously.

5) Final Form (Coupled Nonlinear System)

Often we isolate $\ddot{x}$ and $\ddot{\theta}$. From the above, we can rewrite:

$$(M+m)\,\ddot{x} \;+\; b\,\dot{x} \;+\; m\,\ell\,\sin(\theta)\,\dot{\theta}^2 \;-\; m\,\ell\,\cos(\theta)\,\ddot{\theta} = \; F, \tag{1}$$

and solving for $\ddot{x}$ and $\ddot{\theta}$ gives the expressions often used in simulation:

$$\ddot{x} = \frac{F + m\,\sin(\theta)\,\big(\ell\,\dot{\theta}^2 + g\,\cos(\theta)\big) - b\,\dot{x}}{M + m\,\sin^2(\theta)}, \tag{1}$$

$$\ddot{\theta} = \frac{-F\,\cos(\theta) - m\,\ell\,\dot{\theta}^2\,\cos(\theta)\,\sin(\theta) - (M+m)\,g\,\sin(\theta) + b\,\dot{x}\,\cos(\theta)}{\ell\,\big(M + m\,\sin^2(\theta)\big)}. \tag{2}$$

These two coupled nonlinear equations comprise the principal cart-pole dynamics as used in my Julia code.

Including a Small Inertia or Other Terms

In some implementations (e.g., lines with `I = 0.006` in the code), one may add a moment of inertia for the pendulum about its pivot or other small friction terms. The approach is similar, but typically we either incorporate it via the total ℓ-dependent term or use a separate inertia. In any case, the final structure remains the same.

3.3 Model Assumptions

- The pivot between the pendulum and cart is frictionless (no torque friction).
- The damping b is linear, and no other friction terms are present.
- The pendulum rod is assumed massless (point-mass at the bob).
- Gravity is constant: $g \approx 9.81\,\mathrm{m/s^2}$.
- The cart moves only horizontally in a single dimension.

4 Control Methods and Their Assumptions

4.1 PID Control

A straightforward approach is to control the cart's position x. I define a position error:

$$e(t) = x_{\mathrm{des}} - x(t),$$

and apply a PID law:

$$F_{\mathrm{PID}}(t) = K_p\, e(t) + K_i \int_0^t e(\tau)\, d\tau - K_d\, \dot{x}(t),$$

where K_p, K_i, and K_d are gains chosen by trial and error or by standard PID tuning methods.

PID Assumptions.

- The pendulum remains near upright ($\theta \approx \pi$) so that focusing on x control indirectly keeps the pendulum from tipping.
- The system has enough actuator force F to correct position errors without saturating.
- Large integral gains can cause windup if the error is large or if the pendulum deviates significantly from upright.

4.2 LQR Control

To systematically balance near $\theta = \pi$, I can linearize the system around that point. Let me define small deviations, e.g. $\tilde{\theta} = \theta - \pi$. Then the linearized dynamics become:

$$\dot{\mathbf{x}} = A\,\mathbf{x} + B\,u, \quad u = F,$$

where a typical choice for the cart-pole is:

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{b}{M} & \frac{m\,g}{M} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{b}{M\ell} & \frac{(M+m)\,g}{M\ell} & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 0 \\ \frac{1}{M} \\ 0 \\ \frac{1}{M\ell} \end{pmatrix}.$$

I then define a cost function

$$J = \int_0^\infty \left[\mathbf{x}^\top Q\, \mathbf{x} + u^\top R\, u \right] dt,$$

for some positive-semidefinite Q and positive R, and solve for the LQR gain K_{lqr}. The control is

$$F_{\text{LQR}}(t) = -\, K_{\text{lqr}}\, \mathbf{x}(t).$$

LQR Assumptions.

- The pendulum does not deviate too far from $\theta = \pi$, so the linear approximation holds.
- The damping b and other parameters remain close to nominal values (no large parameter uncertainties).
- No actuator saturation or constraints beyond the linear model near equilibrium.

5 Algorithmic Flow of the Simulation

5.1 Detailed Function-Call Flowchart

Below is an additional flowchart illustrating *how the main functions call each other sequentially* in response to user input from the GUI elements (menus, sliders, buttons, etc.). Each block corresponds to a key function in the code:
In practice:

1. `create_makie_gui()` constructs all GUI elements.
2. Within that GUI, `init_cart()` is called to set defaults.
3. When the user chooses LQR parameters, `compute_lqr_gain_custom()` is invoked.
4. Clicking `Start Simulation` calls `simulate_cartpole()`, which uses `cartpole_dynamics!()` inside the ODE solver.
5. During the animation loop, `update_visualization!()` is called to redraw the cart-pole state.

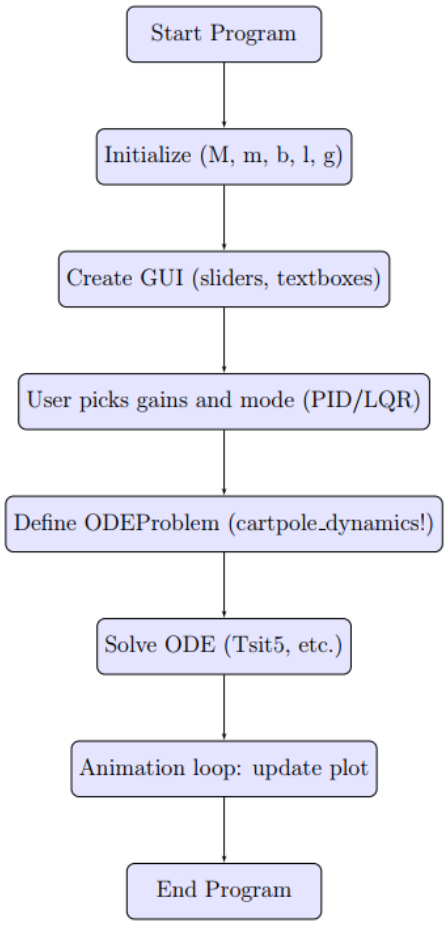

Figure 2: Conceptual flowchart of cart-pole simulation in Julia.

5.2 User Experience Flow Diagram

Here is a step-by-step explanation of Figure 3:

1. **Start GUI**: The user is presented with the interface.
2. **Control Mode Selected?**:
   - *No*: Goes to Error (must pick Open or Closed).
   - *Yes*: Proceeds to *Open or Closed?*.
3. **Open or Closed?**:
   - *Open*: Force $F = 0$, skip advanced control. Flow goes directly to the ODE setup.
   - *Closed*: Must pick either PID or LQR next.
4. **PID or LQR?**:
   - *No Selection*: Leads to an error block (must pick one).
   - *PID*: Use K_p, K_i, K_d.

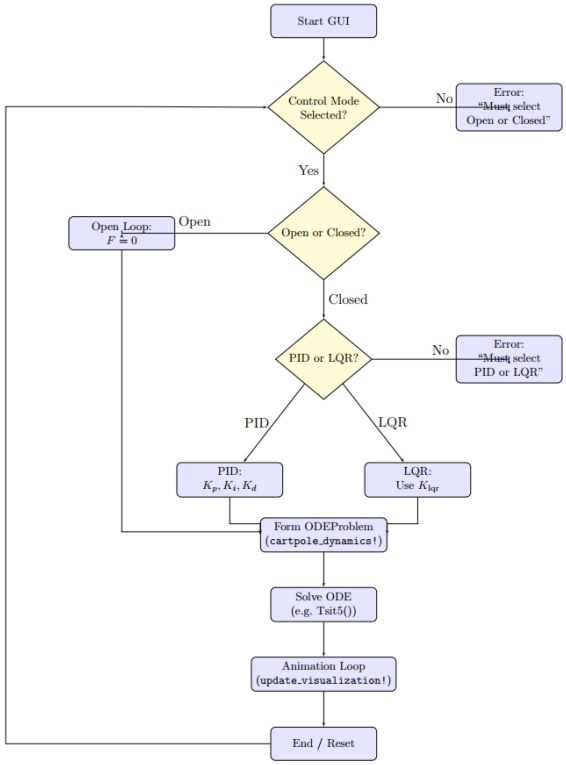

Figure 3: User-experience flow diagram of cart-pole simulation.

- *LQR*: Use K_{lqr}.

5. **Form ODEProblem** and **Solve ODE**: After we know the control scheme, we build the ODE with the chosen control logic. We solve it (e.g. Tsit5()) over a desired time span.

6. **Animation Loop**: The GUI continuously reads solution samples, updating the cart-pole drawing.

7. **End / Reset**: The simulation can end or be reset if the user changes parameters and restarts.

6 Simulation Results

I tested both PID and LQR controllers under various initial angles and desired positions. Some main findings:

- PID works well for small perturbations around upright. However, large integral gains can cause integrator windup, and very large angle deviations may destabilize the system.

- LQR provides a smooth response if the initial state is close to $\theta = \pi$. By adjusting Q and R, I can tune how aggressively the controller penalizes angular deviations or cart movements.

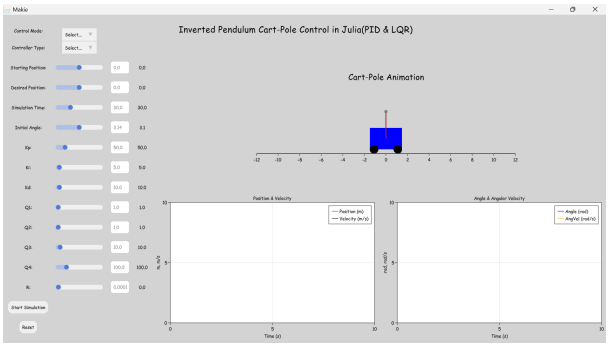

Figure 4: The full GUI of the Cart-Pole simulation in Julia, with sliders, dropdown menus, plots, and a real-time animation area.

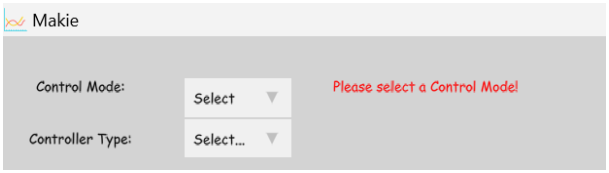

Figure 5: If the Control Mode is not selected, the GUI prompts the user to choose Open or Closed Loop.

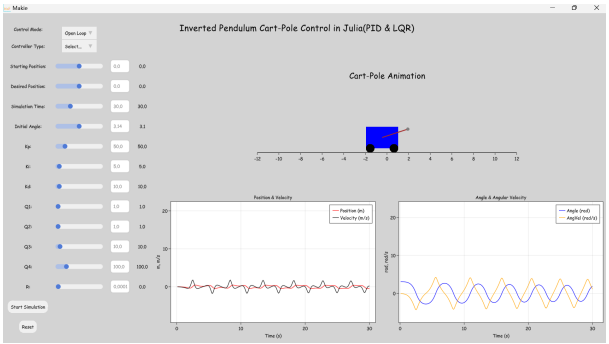

Figure 6: Open-loop mode: no active control force. The pendulum swings freely while the cart remains at rest (or moves under initial conditions).

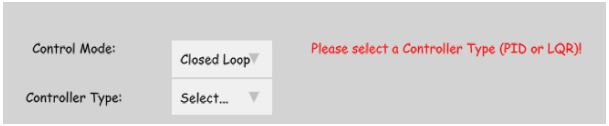

Figure 7: Closed-loop is selected, but the specific controller (PID or LQR) has not been chosen; hence, the GUI requests a valid selection.

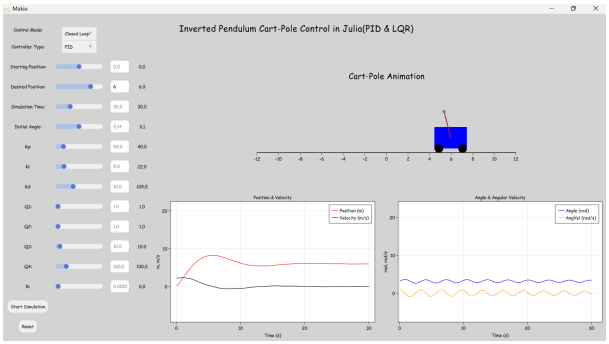

Figure 8: PID controller example: the cart is moved to a positive desired position, and the pendulum remains upright with minor oscillations.

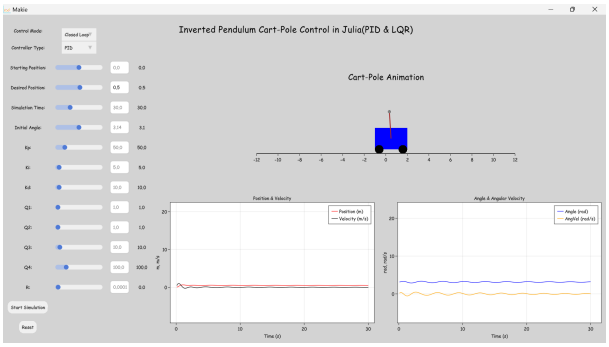

Figure 9: Another PID scenario with different gains. The plots show how position and velocity converge, while the angle stabilizes around the upright equilibrium.

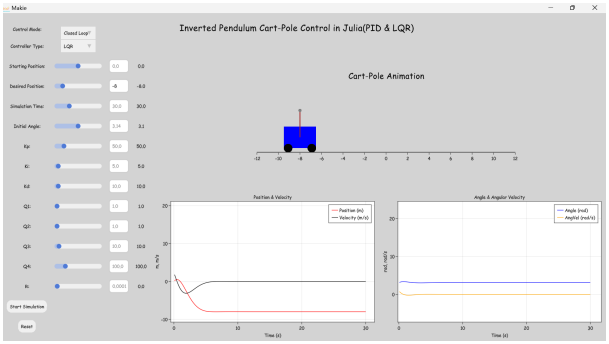

Figure 10: LQR controller moving the cart to a negative position. The pendulum is balanced around $\theta = \pi$ with minimal oscillation.

Comparison of PID and LQR. PID is typically simpler to implement but may require trial-and-error gain tuning, and can be sensitive to large deviations where linear approximations no longer hold. In contrast, LQR uses a more systematic gain computation based on linearization around the upright equilibrium. Once linearized, LQR gains are found by solving a Riccati equation, which generally provides smoother and more robust performance near $\theta = \pi$. However, LQR also assumes that the system remains close to the operating point. If the pendulum is far from upright or if parameters deviate significantly, the linear design may be insufficient without modifications such as gain

scheduling or nonlinear control. Overall, LQR can provide a more elegant solution with fewer tuning knobs compared to PID, but PID remains a practical option if the pendulum does not move too far from equilibrium and careful tuning is performed.

7 Conclusions

I have demonstrated how a **Julia**-based educational toolbox can be created to visualize and control the inverted pendulum on a cart. By integrating `Makie.jl` for interactive visualization, `DifferentialEquations.jl` for ODE solving, and simple UI frameworks (like `GTK` or `Pluto.jl`) for user interaction, it becomes much easier for learners to experiment with real-time adjustments.

Within this toolbox, a PID controller can be used to control the cart position and indirectly balance the pendulum (effectively near the upright position), while an LQR design based on linearization around $\theta = \pi$ provides robust local stabilization with fewer tuning parameters, assuming small initial deviations.

Future Work

- Make a stand-alone application for a simpler user experience.
- Modify the UI for better usability.
- Add advanced control like MPC.
- Implement a swing-up strategy if the pendulum starts near $\theta = 0$.
- Investigate nonlinear control methods that do not rely on the small-angle approximation around $\theta = \pi$.

8 References

References

[1] K. J. Åström and B. Wittenmark, *Computer Controlled Systems: Theory and Design*. Prentice-Hall, 1997.

[2] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*. Pearson, 2014.

[3] Julia Documentation (including `DifferentialEquations.jl` and `ControlSystems.jl`): `https://docs.julialang.org`, accessed 2025.

[4] Inverted Pendulum - System Modeling, Control Tutorials for MATLAB and Simulink (CTMS), ctms.engin.umich.edu, accessed 2025.

[5] Carvales, *Inverted Pendulum* in control-systems-visu-tools, GitHub, accessed 2025.

[6] *Makie.jl: Explanations / Blocks*, docs.makie.org/v0.21/explanations/blocks, accessed 2025.

Appendix

Cart-Pole Simulation GUI: User Manual

(By Melaku Yemaneberhan MESIHU)

1 Introduction

This document provides a brief user manual for the **Cart-Pole Simulation GUI** written in Julia. The GUI demonstrates how to control an inverted pendulum on a moving cart using two main control strategies: **PID** and **LQR**, as well as an **Open Loop** mode.

2 Requirements

- **Julia 1.10.7**
- A project environment with the necessary packages:
  - `GLMakie`, `Observables`, `DifferentialEquations`, `ControlSystems`, `LinearAlgebra`, `GeometryBasics`.
- The **Makie.jl** package must be version **v0.15**. If you have a different version installed and encounter errors, please remove it and install v0.15 specifically:

```
# In the Julia REPL:

] rm Makie              # removes any existing (possibly newer) version
] add Makie@0.15.0      # installs Makie 0.15.0
```

- A **Project.toml** and **Manifest.toml** set up so the environment can be instantiated.

3 Launching the GUI

1. Open a terminal or Julia REPL inside the project folder that contains `Project.toml` and `Manifest.toml`.
2. Activate and instantiate the project environment:

```
julia> ]
pkg> activate .
pkg> instantiate
```

3. Load and run the main script (`cartpole.jl`) that contains the `main()` function:

```
julia> include("cartpole.jl")
```

4. The GUI should automatically appear in a separate window, or within your current plotting environment (e.g., if using VSCode).

4 GUI Layout and Controls

4.1 Control Panel (Left Side)

- **Control Mode Menu:**
  - `Open Loop`: No active control; the cart will not respond to disturbances.
  - `Closed Loop`: Enables either PID or LQR control.
- **Controller Type Menu:**
  - `PID`: Uses proportional-integral-derivative control to move the cart to the desired position and keep the pendulum inverted.
  - `LQR`: Uses the linear-quadratic regulator approach.
- **Parameter Sliders and Textboxes** for:
  - `Starting Position`: Initial horizontal position of the cart.
  - `Desired Position`: Target position of the cart.
  - `Simulation Time`: How long (in seconds) the simulation should run.
  - `Initial Angle`: Starting pendulum angle (in radians).
- **PID Gains** (`Kp`, `Ki`, `Kd`): Adjust these if using PID control.
- **LQR Weights** (`Q1`, `Q2`, `Q3`, `Q4`, `R`): Adjust these if using LQR control.
- **Start Simulation Button**: Begins the simulation with the current parameter settings.
- **Reset Button**: Resets all parameters to their default values and clears the plots.
- **Error Label**: Displays warnings or prompts if a required selection is missing.

4.2 Visualization Panel (Right Side)

- **Cart-Pole Animation**: Shows the cart, wheels, and pendulum. Position and angle will update in real time.
- **Position & Velocity Plot**: Plots the cart's position and velocity over time.
- **Angle & Angular Velocity Plot**: Plots the pendulum's angle and angular velocity over time.

5 Using the GUI: Step-by-Step

1. **Select the Control Mode**:
   - `Open Loop` for a no-control demonstration.
   - `Closed Loop` to enable PID or LQR.
2. If `Closed Loop` is selected, **choose `PID` or `LQR`** in the *Controller Type* menu.

3. Adjust any of the **sliders/textboxes** for starting or desired positions, simulation time, or initial angle.

4. If using `PID`, refine `Kp`, `Ki`, `Kd` as desired. If using `LQR`, adjust the `Q` and `R` values to test different weightings.

5. Click the **Start Simulation** button to run. The animation and plots will update over the chosen simulation time.

6. **Observe and interpret the results** in real-time:

   - The left *Position & Velocity* plot shows how the cart moves.
   - The right *Angle & Angular Velocity* plot shows how well the pendulum is stabilized (ideally near π radians for upright).

7. To rerun under new conditions, either change parameters and press *Start Simulation* again or use the *Reset* button to return everything to defaults.

6 Notes

- If you select `Closed Loop` but do not pick `PID` or `LQR`, the GUI will display an error message.
- Changing parameters **during** a simulation does not affect the current run. It will take effect only on the next run.
- Use the **Reset Button** if you need to return all parameters to standard presets.
- **Makie Compatibility:** This simulation GUI was developed and tested on **Makie v0.15.0**. Make sure you install exactly `Makie@0.15.0` if you encounter issues with other versions.
- After successfully running everything, you might encounter an unresponsive GUI page. This is because everyone has different screen and this is normal; feel free to adjust some figure or font sizes in the script as needed.

7 Conclusion

This short manual explains how to launch and operate the Cart-Pole GUI. By experimenting with different parameters, you can gain insight into control strategies for an inverted pendulum system. For more advanced usage, feel free to modify the `Project.toml`, explore alternative ODE solvers, or further customize the code.