Aim

To simulate and visualize the behavior of a PID-controlled train attempting to catch a falling cube on an inclined rail.

Objective

- To implement a PID controller for adjusting the position of a train on an inclined rail.
- To simulate the motion of a falling cube and the train's attempt to catch it.
- To visualize the system's response and performance through animations and plots.

Summary

This project involves simulating a PID-controlled train on an inclined rail trying to catch a falling cube. The simulation calculates the train's displacement, velocity, acceleration, and PID controller's error metrics (horizontal error, rate of change of error, and integral of error) over time. An animation is generated to visualize the train and cube movements, and performance metrics are plotted to assess the PID controller's effectiveness.

Tools and Libraries Used

- **Python Libraries**:
    - matplotlib (for plotting and animation)
    - numpy (for numerical calculations)
    - random (for generating random initial positions)

Procedure

1. **Initialization**: Set initial parameters including the number of trials, incline angle, gravitational constant, and mass of the train. Define PID constants KpK_pKp, KdK_dKd, and KiK_iKi.
2. **Generate Random Positions**: Define a function to generate random initial positions for the falling cube.
3. **Simulation Setup**: Initialize arrays to store displacement, velocity, acceleration, and error metrics for the train and cube over time.
4. **PID Control Implementation**:
    - For each time step, calculate the horizontal error between the train's position and the falling cube's position.
    - Update the error metrics (error derivative and integral).
    - Compute the control force using the PID formula.
    - Update the train's acceleration, velocity, and displacement based on the control force and gravitational component.

- Check if the train catches the cube within a given tolerance.

5. **Animation and Plotting**:
    - Define a function to update the plot with current positions and metrics.
    - Create subplots to visualize the displacement, velocity, acceleration, and error metrics.
    - Generate and display the animation showing the train and cube movements and control performance.

CODE:

```
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import matplotlib.animation as animation
import numpy as np
import random

################################################################################

# ONLY CHANGE THESE INPUTS FOR THE CODE TO WORK PROPERLY

# Initialize input values
trials=3
incl_angle=np.pi/6*1 # Keep the angle between 0 and +pi/6 radians
g=10
mass_cart=100 # [kg]

# Tune the constants
K_p=300
K_d=300
K_i=10
################################################################################
```

```
trials_global=trials

# Generate random x-positions for a falling cube
def set_x_ref(incl_angle):
    rand_h=random.uniform(0,120)
    rand_v=random.uniform(20+120*np.tan(incl_angle)+6.5,40+120*np.tan(incl_angle)+6.5)
    return rand_h,rand_v

dt=0.02
t0=0
t_end=5
t=np.arange(t0,t_end+dt,dt)

F_g=-mass_cart*g

displ_rail=np.zeros((trials,len(t)))
v_rail=np.zeros((trials,len(t)))
a_rail=np.zeros((trials,len(t)))
pos_x_train=np.zeros((trials,len(t)))
pos_y_train=np.zeros((trials,len(t)))
e=np.zeros((trials,len(t)))
e_dot=np.zeros((trials,len(t)))
e_int=np.zeros((trials,len(t)))

pos_x_cube=np.zeros((trials,len(t)))
pos_y_cube=np.zeros((trials,len(t)))

F_ga_t=F_g*np.sin(incl_angle) # Tangential component of the gravity force
```

```
init_pos_x=120
init_pos_y=120*np.tan(incl_angle)+6.5
init_displ_rail=(init_pos_x**2+init_pos_y**2)**(0.5)
init_vel_rail=0
init_a_rail=0

init_pos_x_global=init_pos_x # Used for determining the dimensions of the animation window.

trials_magn=trials
history=np.ones(trials)
while(trials>0): # Determines how many times cube falls down
    pos_x_cube_ref=set_x_ref(incl_angle)[0] # Cube's initial x position
    pos_y_cube_ref=set_x_ref(incl_angle)[1] # Cube's initial y position
    times=trials_magn-trials
    pos_x_cube[times]=pos_x_cube_ref
    pos_y_cube[times]=pos_y_cube_ref-g/2*t**2
    win=False
    delta=1

    # Implement PID for the train position
    for i in range(1,len(t)):
        # Insert the initial values into the beginning of the predefined arrays.
        if i==1:
            displ_rail[times][0]=init_displ_rail
            pos_x_train[times][0]=init_pos_x
            pos_y_train[times][0]=init_pos_y
            v_rail[times][0]=init_vel_rail
            a_rail[times][0]=init_a_rail
```

```
    # Compute the horizontal error
    e[times][i-1]=pos_x_cube_ref-pos_x_train[times][i-1]

    if i>1:
       e_dot[times][i-1]=(e[times][i-1]-e[times][i-2])/dt
       e_int[times][i-1]=e_int[times][i-2]+(e[times][i-2]+e[times][i-1])/2*dt
    if i==len(t)-1:
       e[times][-1]=e[times][-2]
       e_dot[times][-1]=e_dot[times][-2]
       e_int[times][-1]=e_int[times][-2]

    F_a=K_p*e[times][i-1]+K_d*e_dot[times][i-1]+K_i*e_int[times][i-1]
    F_net=F_a+F_ga_t
    a_rail[times][i]=F_net/mass_cart
    v_rail[times][i]=v_rail[times][i-1]+(a_rail[times][i-1]+a_rail[times][i])/2*dt
    displ_rail[times][i]=displ_rail[times][i-1]+(v_rail[times][i-1]+v_rail[times][i])/2*dt
    pos_x_train[times][i]=displ_rail[times][i]*np.cos(incl_angle)
    pos_y_train[times][i]=displ_rail[times][i]*np.sin(incl_angle)+6.5

    # Try to catch it
    if (pos_x_train[times][i]-5<pos_x_cube[times][i]+3 and
pos_x_train[times][i]+5>pos_x_cube[times][i]-3) or win==True:
       if (pos_y_train[times][i]+3<pos_y_cube[times][i]-2 and
pos_y_train[times][i]+8>pos_y_cube[times][i]+2) or win==True:
          win=True
          if delta==1:
             change=pos_x_train[times][i]-pos_x_cube[times][i]
             delta=0
          pos_x_cube[times][i]=pos_x_train[times][i]-change
```

```
            pos_y_cube[times][i]=pos_y_train[times][i]+5


    init_displ_rail=displ_rail[times][-1]
    init_pos_x=pos_x_train[times][-1]+v_rail[times][-1]*np.cos(incl_angle)*dt
    init_pos_y=pos_y_train[times][-1]+v_rail[times][-1]*np.sin(incl_angle)*dt
    init_vel_rail=v_rail[times][-1]
    init_a_rail=a_rail[times][-1]
    history[times]=delta
    trials=trials-1

################################ ANIMATION ###################################
len_t=len(t)
frame_amount=len(t)*trials_global
def update_plot(num):

    platform.set_data([pos_x_train[int(num/len_t)][num-int(num/len_t)*len_t]-3.1,\
    pos_x_train[int(num/len_t)][num-int(num/len_t)*len_t]+3.1],\
    [pos_y_train[int(num/len_t)][num-int(num/len_t)*len_t],\
    pos_y_train[int(num/len_t)][num-int(num/len_t)*len_t]])

    cube.set_data([pos_x_cube[int(num/len_t)][num-int(num/len_t)*len_t]-1,\
    pos_x_cube[int(num/len_t)][num-int(num/len_t)*len_t]+1],\
    [pos_y_cube[int(num/len_t)][num-int(num/len_t)*len_t],\
    pos_y_cube[int(num/len_t)][num-int(num/len_t)*len_t]])

    if trials_magn*len_t==num+1 and num>0: # All attempts must be successful
        if sum(history)==0:
            success.set_text('CONGRATS! YOU DID IT!')
```

```
        else:
            again.set_text('DON’T GIVE UP! YOU CAN DO IT!')

    displ_rail_f.set_data(t[0:(num-int(num/len_t)*len_t)],
        displ_rail[int(num/len_t)][0:(num-int(num/len_t)*len_t)])

    v_rail_f.set_data(t[0:(num-int(num/len_t)*len_t)],
        v_rail[int(num/len_t)][0:(num-int(num/len_t)*len_t)])

    a_rail_f.set_data(t[0:(num-int(num/len_t)*len_t)],
        a_rail[int(num/len_t)][0:(num-int(num/len_t)*len_t)])

    e_f.set_data(t[0:(num-int(num/len_t)*len_t)],
        e[int(num/len_t)][0:(num-int(num/len_t)*len_t)])

    e_dot_f.set_data(t[0:(num-int(num/len_t)*len_t)],
        e_dot[int(num/len_t)][0:(num-int(num/len_t)*len_t)])

    e_int_f.set_data(t[0:(num-int(num/len_t)*len_t)],
        e_int[int(num/len_t)][0:(num-int(num/len_t)*len_t)])

    return displ_rail_f,v_rail_f,a_rail_f,e_f,e_dot_f,e_int_f,platform,cube,success,again

fig=plt.figure(figsize=(16,9),dpi=120,facecolor=(0.8,0.8,0.8))
gs=gridspec.GridSpec(4,3)

# Create main window
ax_main=fig.add_subplot(gs[0:3,0:2],facecolor=(0.9,0.9,0.9))
plt.xlim(0,init_pos_x_global)
```

```
plt.ylim(0,init_pos_x_global)
plt.xticks(np.arange(0,init_pos_x_global+1,10))
plt.yticks(np.arange(0,init_pos_x_global+1,10))
plt.grid(True)

copyright=ax_main.text(0,122,'© Mark Misin Engineering',size=12)

rail=ax_main.plot([0,init_pos_x_global],[5,init_pos_x_global*np.tan(incl_angle)+5],'k',linewidth=6)
platform,=ax_main.plot([],[],'b',linewidth=18)
cube,=ax_main.plot([],[],'k',linewidth=14)

bbox_props_success=dict(boxstyle='square',fc=(0.9,0.9,0.9),ec='g',lw='1')
success=ax_main.text(40,60,'',size='20',color='g',bbox=bbox_props_success)

bbox_props_again=dict(boxstyle='square',fc=(0.9,0.9,0.9),ec='r',lw='1')
again=ax_main.text(30,60,'',size='20',color='r',bbox=bbox_props_again)

# Plot windows
ax1v=fig.add_subplot(gs[0,2],facecolor=(0.9,0.9,0.9))
displ_rail_f,=ax1v.plot([],[],'-b',linewidth=2,label='displ. on rails [m]')
plt.xlim(t0,t_end)
plt.ylim(np.min(displ_rail)-abs(np.min(displ_rail))*0.1,np.max(displ_rail)+abs(np.max(displ_rail))*0.1)
plt.grid(True)
plt.legend(loc='lower left',fontsize='small')

ax2v=fig.add_subplot(gs[1,2],facecolor=(0.9,0.9,0.9))
v_rail_f,=ax2v.plot([],[],'-b',linewidth=2,label='velocity on rails [m/s]')
plt.xlim(t0,t_end)
plt.ylim(np.min(v_rail)-abs(np.min(v_rail))*0.1,np.max(v_rail)+abs(np.max(v_rail))*0.1)
```

```
plt.grid(True)
plt.legend(loc='lower left',fontsize='small')

ax3v=fig.add_subplot(gs[2,2],facecolor=(0.9,0.9,0.9))
a_rail_f,=ax3v.plot([],[],'-b',linewidth=2,label='accel. on rails [m/s^2] = F_net/m_platf.')
plt.xlim(t0,t_end)
plt.ylim(np.min(a_rail)-abs(np.min(a_rail))*0.1,np.max(a_rail)+abs(np.max(a_rail))*0.1)
plt.grid(True)
plt.legend(loc='lower left',fontsize='small')

ax1h=fig.add_subplot(gs[3,0],facecolor=(0.9,0.9,0.9))
e_f,=ax1h.plot([],[],'-b',linewidth=2,label='horizontal error [m]')
plt.xlim(t0,t_end)
plt.ylim(np.min(e)-abs(np.min(e))*0.1,np.max(e)+abs(np.max(e))*0.1)
plt.grid(True)
plt.legend(loc='lower left',fontsize='small')

ax2h=fig.add_subplot(gs[3,1],facecolor=(0.9,0.9,0.9))
e_dot_f,=ax2h.plot([],[],'-b',linewidth=2,label='change of horiz. error [m/s]')
plt.xlim(t0,t_end)
plt.ylim(np.min(e_dot)-abs(np.min(e_dot))*0.1,np.max(e_dot)+abs(np.max(e_dot))*0.1)
plt.grid(True)
plt.legend(loc='lower left',fontsize='small')

ax3h=fig.add_subplot(gs[3,2],facecolor=(0.9,0.9,0.9))
e_int_f,=ax3h.plot([],[],'-b',linewidth=2,label='sum of horiz. error [m*s]')
plt.xlim(t0,t_end)
plt.ylim(np.min(e_int)-abs(np.min(e_int))*0.1,np.max(e_int)+abs(np.max(e_int))*0.1)
plt.grid(True)
```

```
plt.legend(loc='lower left',fontsize='small')


pid_ani=animation.FuncAnimation(fig,update_plot,
   frames=frame_amount,interval=20,repeat=False,blit=True)
plt.show()
```

Highlights

- **PID Control**: Demonstrates how PID control can be used to adjust the position of a train to track a moving target.
- **Real-Time Visualization**: Uses animation to provide a dynamic visualization of the system's response.
- **Error Metrics**: Plots horizontal error, its rate of change, and its integral to evaluate the controller's performance.

Conclusion

The simulation effectively demonstrates the application of PID control to a dynamic system involving a train and a falling cube. The animated visualization provides insight into the system's behavior and the effectiveness of the PID controller in adjusting the train's position. The PID parameters can be fine-tuned to improve performance, and the plots of error metrics offer valuable feedback for optimizing the control strategy.