# VREP倒立摆使用文档

## 1 场景

找到文件invertedPendulum打开即可见到如下画面。点击运行，倒立摆开始仿真。

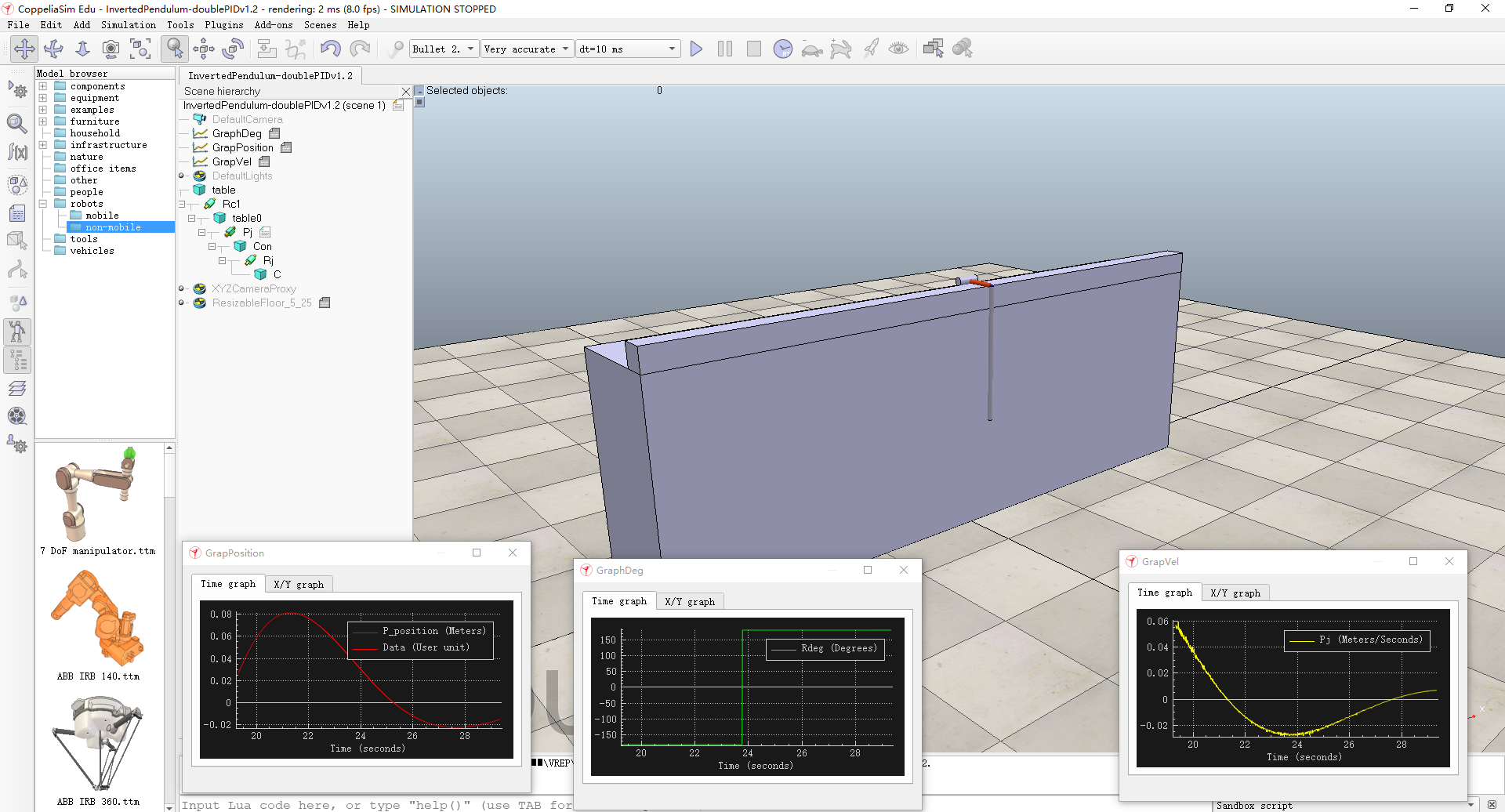


图1-倒立摆场景

## 2 控制

### 2.1 并联PID控制

倒立摆有两个控制量。小车的位移量和摆杆的角度。由于小车在有固定距离的导轨上运动，为了防止小车撞上限位端，所以需要控制小车的位移量。倒立摆的摆杆要进入稳定状态就需要控制摆杆的角度。

这里使用并联PID分别控制两个量，最后把两个PID输出量叠加作为执行电机的控制量。如下图所示。

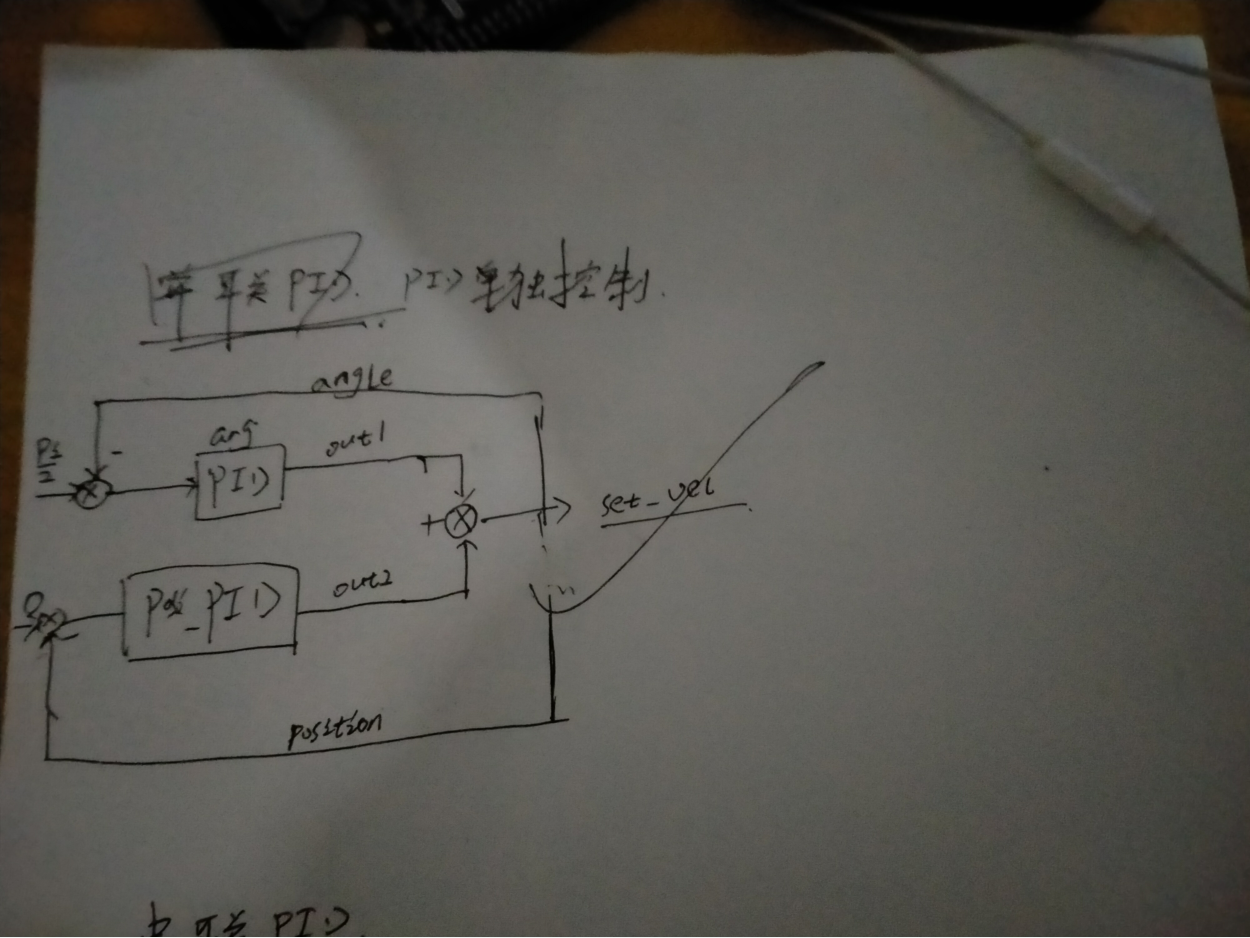


图2-并联PID控制框图

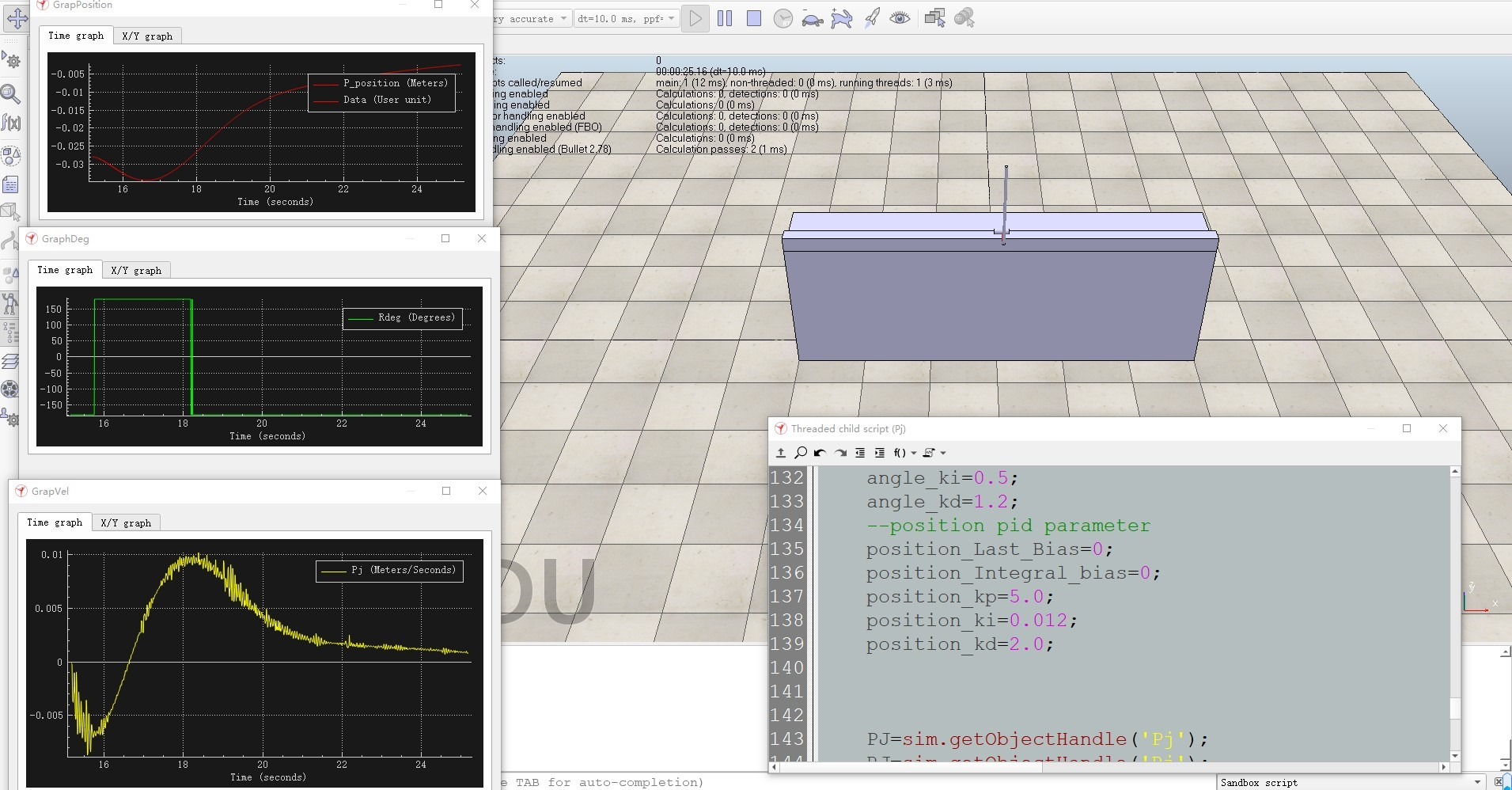


图3-倒立摆稳定状态

### 2.2 并联PID调参

并联PID参数调节和单PID参数调节一样。

## 3 代码

控制倒立摆代码采用Lua编写。

### 3.1 PID控制

#### 3.1.1 位置控制

--position pid controler

position\_pid=function(Pencoder,Ptarget)

--Bias

Bias=Pencoder-Ptarget;

--intergal bias

position\_Integral\_bias=position\_Integral\_bias+Bias;

--output

output=position\_kp\*Bias+position\_ki\*position\_Integral\_bias+position\_kd\*(Bias-position\_Last\_Bias);

--Last\_Bias

position\_Last\_Bias=Bias;

--result=sim.auxiliaryConsolePrint(consoleHandle,text);

--sim.auxiliaryConsolePrint(pidconsoleHandle,Bias);

return output;

end

#### 3.1.2 角度控制

--angle pid controler

angle\_pid=function(Rencoder,Rtarget)

--Bias

Bias=Rencoder-Rtarget;

print("Bias: ",Bias,"\n");

--intergal bias

angle\_Integral\_bias=angle\_Integral\_bias+Bias;

--output

output=angle\_kp\*Bias+angle\_ki\*angle\_Integral\_bias+angle\_kd\*(Bias-angle\_Last\_Bias);

--Last\_Bias

angle\_Last\_Bias=Bias;

--result=sim.auxiliaryConsolePrint(consoleHandle,text);

--sim.auxiliaryConsolePrint(pidconsoleHandle,Bias);

return output;

end

### 3.2 能量起摆

--[[

energy swing

--]]

energySwing=function()

--energy swing

while 1 do

--Pdeg=sim.getJointPosition(PJ);

Rdeg=sim.getJointPosition(RJ);

if (Rdeg<=160/180\*pi and Rdeg>0) then

--sim.setJointTargetPosition(PJ,-0.1);

sim.setJointTargetVelocity(PJ,-0.25);

Sleep(Rdeg/10);

elseif (Rdeg>=-160/180\*pi and Rdeg<=0) then

--sim.setJointTargetPosition(PJ,0.1);

sim.setJointTargetVelocity(PJ,0.25);

Sleep(Rdeg/10);

else

break;

end

--print("Rdeg: ",Rdeg,"\n");

end

end

### 3.3 小角度稳摆

--stable

stable=function()

while 1 do

Rdeg=sim.getJointPosition(RJ);

Pdeg=sim.getJointPosition(PJ);

--position pid

--print("Pdeg: ",Pdeg,"\n");

P\_out=position\_pid(Pdeg,0.0);

--P\_out=limitOutput(P\_out);

--angle pid

由于倒立摆摆杆在180°临界时有两种情形。

1.摆杆从负角度范围到-180°

2.摆杆从正角度范围到+180°

但是，-180°和+180°都代表同一个位置。因此在角度PID调用时需要考虑摆杆从那个方向旋转。最后还需要将输出值取反，才可以让输出值对倒立摆角度值有实际的控制作用。

**if (Rdeg<0) then**

**A\_out=-angle\_pid(Rdeg,-pi); --- +**

**else**

**A\_out=-angle\_pid(Rdeg,pi); ---- -**

**End**

**该位置将两个PID输出效果叠加。**

out=P\_out+A\_out;

--A\_out=angle\_pid(Rdeg,P\_out);

**最后进行限幅操作**

out=limitOutput(out);

--print("A\_out: ",A\_out,"\n");

--sim.setJointTargetPosition(PJ,A\_out);

**对电机的速度进行控制**

sim.setJointTargetVelocity(PJ,out);

--Sleep(0.1)

--if fail

if((Rdeg<=50/180\*pi and Rdeg>=-50/180\*pi)) then

break;

end

end

end

### 3.4 其它

#### 3.4.1 延时

function Sleep(n)

local t0 = os.clock()

while os.clock() - t0 <= n do end

end

#### 3.4.2 限制幅度

--limit output value

limitOutput=function(output)

if (output>=5.0) then

output=5.0;

elseif(output<=-5.0) then

output=-5.0;

end

return output;

end

#### 3.4.3 主线程

function sysCall\_threadmain()

-- Put some initialization code here

pi=3.1415926;

--angle pid parameter

angle\_Last\_Bias=0;

angle\_Integral\_bias=0;

angle\_kp=28.0;

angle\_ki=0.5;

angle\_kd=1.2;

--position pid parameter

position\_Last\_Bias=0;

position\_Integral\_bias=0;

position\_kp=5.0;

position\_ki=0.012;

position\_kd=2.0;

PJ=sim.getObjectHandle('Pj');

RJ=sim.getObjectHandle('Rj');

Rdeg=sim.getJointPosition(RJ);

Pdeg=sim.getJointPosition(PJ);

--pidconsoleHandle=sim.auxiliaryConsoleOpen("pidConsole",2,2,nil,nil,nil,nil);

--swing

energySwing();

--stable

stable();

end