



课程大作业报告

课程名称: 控制系统仿真

授课教师: 杜秀华老师

学生姓名 : 傅嘉豪

学号 : 5120309341

成绩: _____

2015 年 12 月 14 日

Abstract

倒立摆系统是一个典型的快速、多变量、非线性、强耦合、自然不稳定的高阶系统。研究其控制理论和方法，对于双足机器人行走、火箭发射过程调整和直升机飞行控制领域有重要的现实意义。本文以直轨二阶倒立摆为仿真对象，针对 iOS 平台开发了一款能实时、直观反应系统性能的仿真软件，并可实时调节模型、控制器参数，以分析各个环节参数对倒立摆系统的影响。

常规的倒立摆仿真系统都是基于 PC 下某些软件平台的，能较好地对倒立摆的阶跃响应等给出仿真结果，但分析结果缺乏直观感受，且干扰施加单一，难以反应真实系统所受外界干扰情况。本文则利用智能手机里加速度传感器等模块，真实地采集外界干扰并作用于被仿真系统上，能直观、清晰地反映系统的动态性能，尤其是对各种干扰的抵抗。

本文首先对二阶倒立摆做了简单介绍，而后从力学分析出发，利用牛顿运动学公式，建立二阶倒立摆的数学模型，并整理为状态空间表达式；在小范围内对系统进行线性化处理，根据线性化结果设计 LQR 控制器。

在数学模型和控制器基础上，利用 iOS 平台新一代的 Swift 语言，直观地将二阶倒立摆这一被控对象显示在手机屏幕上，直观地再现了二阶倒立摆的实时运动过程；同时设置了一系列滑条，用以改变整个系统的诸多参数。

最后在上述工作的基础上，改变各个模型及控制器参数，分析了不同参数对系统的影响，验证了 LQR 控制器设计的可靠性，更深入地了解了二阶倒立摆这一经典的被控对象。

关键字：二阶倒立摆 数学模型 Swift 语言 LQR 控制器

Contents

Abstract	I
1 二阶倒立摆系统构成	1
2 倒立摆的数学模型	2
2.1 模型简化	2
2.2 符号说明	2
2.3 系统建模	2
3 iOS 平台实现	5
3.1 App 界面	5
3.2 App 功能	6
3.2.1 更改模型参数	6
3.2.2 调节控制器参数	7
3.2.3 利用重力感应施加干扰	7
3.3 实现原理	8
3.3.1 摆数据 linkage 类	8
3.3.2 系统仿真 twoOrderInvertedPendulum 类	9
3.3.3 示波器曲线 OscilloscopeCurve 类	10
3.3.4 App 主控 ViewController类	11
3.3.5 数学支持 mathSupport.swift 库	13
4 仿真实验	14
4.1 开环分析	14
4.2 闭环分析	14
4.2.1 LQR 控制器设计	14
4.2.2 控制器性能分析	16
4.3 误差分析	16
5 总结	17
Acknowledgements	19
Bibliography	20
Appendix	21

Chapter 1

二阶倒立摆系统构成

二阶倒立摆如图1.1所示，主要有四部分组成：上摆、下摆、小车和直线滑轨。摆如图1.2所示，由摆锤（或轴）和摆杆组成，分别具有质量。

其中直线滑轨固定，小车只能在直线滑轨上做一维直线运动，具有一个自由度 x ；下摆杆下端固定在位于小车几何中点的转轴上，只能在平面内绕转轴转动，具有一个自由度 θ_1 ；上摆杆下端固定在位于下摆杆顶端几何中点的转轴上，只能在平面内绕转轴转动，具有一个自由度 θ_2 。可见，二阶倒立摆是一个三自由度的物理系统。

小车车轮上安装有旋转编码器，用以反馈小车的实时速度和相对于直线轨道中点的距离；摆杆下端也都安装有旋转编码器，用以计算摆杆同竖直方向的夹角。

小车上安装有驱动电机，在电机转矩的控制下产生主动运动；上、下摆杆不收扭矩的控制，只能依靠惯性做旋转运动。因此，二阶倒立摆是一个典型的欠激励系统。

在控制器的作用下，小车电机输出扭矩，推动小车及上下摆杆的运动，抵抗外界干扰，维持上下摆杆处于直立状态。

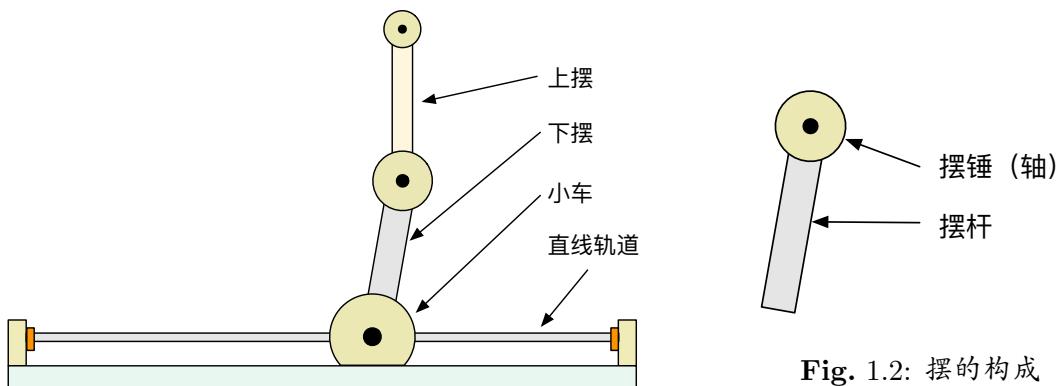


Fig. 1.2: 摆的构成

Fig. 1.1: 二阶倒立摆系统结构

Chapter 2

倒立摆的数学模型

2.1 模型简化

考虑实际模型的复杂性，在误差允许范围内，做以下简化

- (1) 小车、上下摆都是刚体；
- (2) 上下摆的摆锤（轴）不相对于摆杆运动，作为质点处理；
- (3) 上下摆的摆杆为均质细杆，只考虑长度和质量；
- (4) 摆在绕轴运动过程中受到摩擦力矩的作用，大小正比于其转动角速度；
- (5) 小车沿直线轨道运动过程中受到摩擦力的作用，大小正比于小车速度；
- (6) 小车电机输出与控制电压成正比，无死区，无滞后，仅有饱和特性；

2.2 符号说明

本文中所用到的符号如表2.1所示。

2.3 系统建模

分析倒立摆系统各个部件受力，如图2.1所示。

考虑各个转轴上力的方向和大小均不确定，故不宜采用牛顿方法分析。采用 Lagrange 方法建立运动学模型[2]。

取状态向量

$$\Theta = \begin{bmatrix} x \\ \theta_1 \\ \theta_2 \end{bmatrix} \quad (2.1)$$

Table 2.1: 系统中所用到的符号说明

符号	物理量	单位	模型中取值
x	小车位移	m	/
θ_1	下摆较竖直方向偏角	rad	/
θ_2	上摆较竖直方向偏角	rad	/
L_0	直轨长度	m	1.400
m_0	小车质量	kg	1.000
m_1	下摆质量	kg	0.293
J_1	下摆转动惯量	$\text{kg} \cdot \text{m}^2$	0.002569
L_1	下摆杆长度	m	0.3293
l_1	下摆杆质心到转轴距离	m	0.2769
m_2	上摆质量	kg	0.220
J_2	上摆转动惯量	$\text{kg} \cdot \text{m}^2$	0.001227
L_2	上摆杆长度	m	0.2587
l_2	上摆杆质心到转轴距离	m	0.2156
c_0	小车与直轨摩擦系数	$\text{N} \cdot \text{s}/\text{m}$	10.00
c_1	下摆杆与转轴摩擦系数	$\text{N} \cdot \text{m} \cdot \text{s}$	0.00071
c_2	上摆杆与转轴摩擦系数	$\text{N} \cdot \text{m} \cdot \text{s}$	0.00071
g	重力加速度	m/s^2	9.80

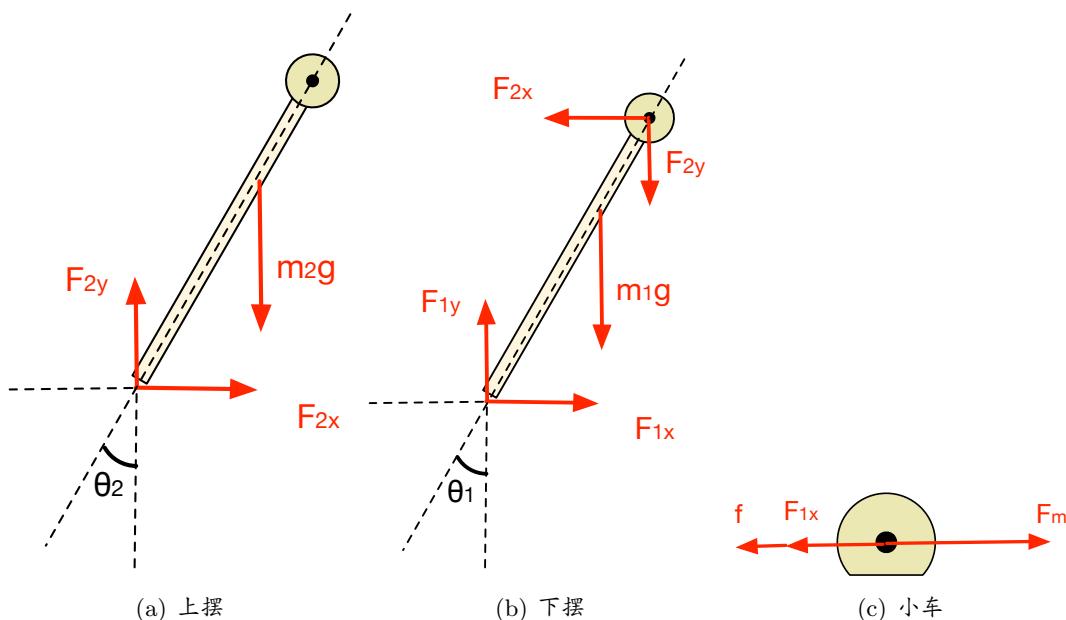


Fig. 2.1: 二阶倒立摆各个关节受力分析

则可得

$$M\ddot{\Theta} + C\dot{\Theta} + G = F \quad (2.2)$$

其中

$$\left\{ \begin{array}{l} M = \begin{bmatrix} m_0 + m_1 + m_2 & (m_1 l_1 + m_2 L_1) \cos \theta_1 & m_2 l_2 \cos \theta_2 \\ (m_1 l_1 + m_2 L_1) \cos \theta_1 & J_1 + m_1 l_1^2 + m_2 L_1^2 & m_2 L_1 l_2 \cos(\theta_1 - \theta_2) \\ m_2 l_2 \cos \theta_2 & m_2 L_1 l_2 \cos(\theta_1 - \theta_2) & J_2 + m_2 l_2^2 \end{bmatrix} \\ C = \begin{bmatrix} c_0 & -(m_1 l_1 + m_2 L_1) \sin \theta_1 \dot{\theta}_1 & -m_2 l_2 \sin \theta_2 \dot{\theta}_2 \\ 0 & c_1 + c_2 & m_2 L_1 l_2 \sin(\theta_1 - \theta_2) \dot{\theta}_2 - c_2 \\ 0 & -m_2 L_1 l_2 \sin(\theta_1 - \theta_2) \dot{\theta}_1 - c_2 & c_2 \end{bmatrix} \\ G = \begin{bmatrix} 0 \\ -(m_1 l_1 + m_2 L_1) g \sin \theta_1 \\ -m_2 g l_2 \sin \theta_2 \end{bmatrix} \\ F = \begin{bmatrix} \tau \\ 0 \\ 0 \end{bmatrix} \end{array} \right. \quad (2.3)$$

为化成状态空间模型，对式2.2变形

$$\ddot{\Theta} = -M^{-1}C\dot{\Theta} + M^{-1}(F - G) \quad (2.4)$$

扩展式2.4为状态空间形式

$$\begin{bmatrix} \dot{\Theta} \\ \ddot{\Theta} \end{bmatrix} = \begin{bmatrix} 0 & I \\ 0 & -M^{-1}C \end{bmatrix} \begin{bmatrix} \Theta \\ \dot{\Theta} \end{bmatrix} + \begin{bmatrix} 0 \\ M^{-1}(F - G) \end{bmatrix} \quad (2.5)$$

该模型为非线性模型。其中 M 、 C 、 G 矩阵都与系统状态变量有关，而 F 矩阵作为系统控制量也在实时变动。故以上矩阵均需要实时计算，宜采用离散相似法进行仿真。

Chapter 3

iOS 平台实现

3.1 App 界面

在 XCode 7.1.1 开发平台上，利用 Swift 语言，编写 app 界面如图3.1所示。

图3.1(b)中，中部的直线轨道、小车和上下摆组成倒立摆系统。如前文 Chapter 1 中所述，小车能在滑轨上做直线运动，同时带动上下摆围绕其各自转轴转动。上方两个按钮分别控制程序的运行/暂停及初始化。

右侧六个滑块分为上下两组，上半部分三个滑块用于控制上摆参数，下半部分三个滑块

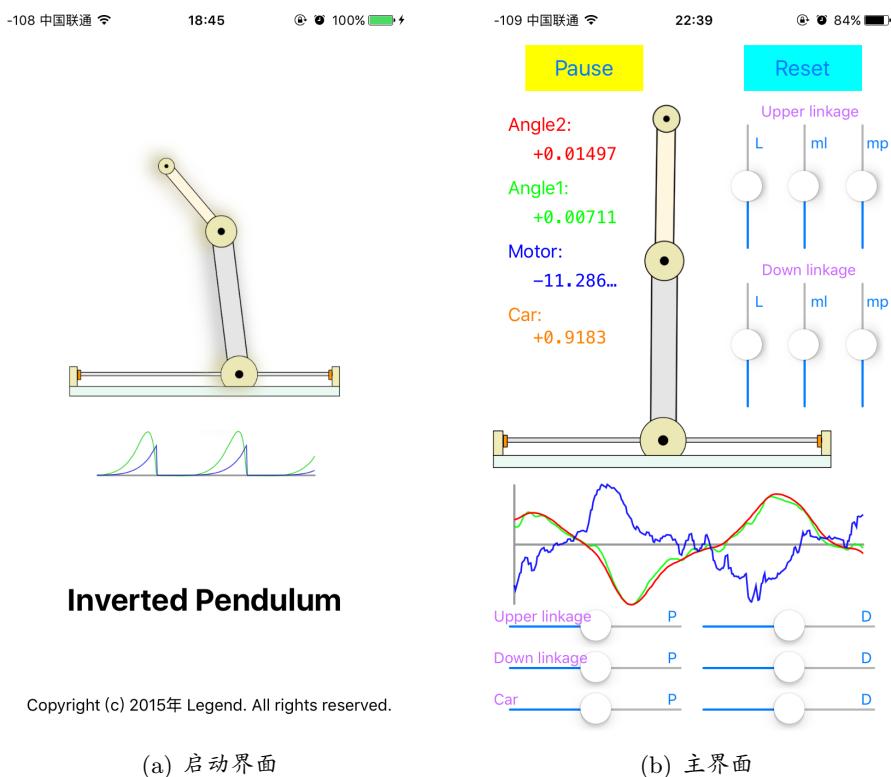


Fig. 3.1: App 界面

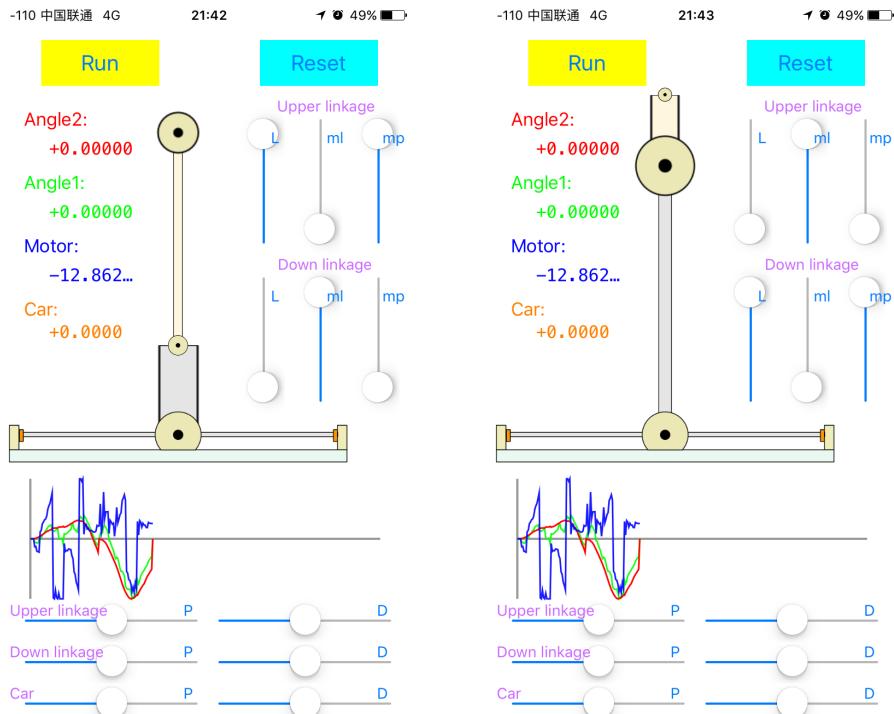


Fig. 3.2: 通过滑块调整模型参数

用于控制下摆参数，调节范围均为表2.1中模型原始数据的 0.5 ~ 1.5 倍。其中最左侧用于调节摆杆长度 L ，不改变摆杆质量等；中间滑块用于调节摆杆质量 m_l ，且摆杆质量正比于图形中摆杆的宽度；右侧滑块用于调节摆锤（轴）的质量 m_p ，且其质量正比于图形中摆锤（轴）的直径。调节上述三个参数都会引起摆重心位置 l_i 和转动惯量 J_i 的变化，进而影响控制效果。

根据状态空间模型，利用 LQR 优化分析，选用 PD 控制器，被控量包括上摆偏角 θ_2 、下摆偏角 θ_1 和小车位置 x 三个参数。下方六个滑块从上到下可分为三组，分别作为控制器中 θ_2 、 θ_1 和 x 反馈控制 PD 系数的放大倍数，调节范围 0 ~ 2。其中左侧三个用以调节比例增益 P 参数，右侧三个用于调节微分增益 D 参数。

直线轨道下方为示波器，显示最近 3.2 s 内上摆偏角 θ_2 （红色）、下摆偏角 θ_1 （绿色）和驱动电机输出（蓝色）的波形，Y 轴方向尺度为自动缩放。摆臂左侧显示当下时刻 θ_2 、 θ_1 、驱动电机输出和 x 的具体数值，角度单位为弧度 rad。

3.2 App 功能

3.2.1 更改模型参数

如3.1中所述，调节右上方六个滑块，可以动态改变倒立摆模型参数，如图3.2所示。在运行过程中，为对模型参数调节效果有直观感受，在手指触碰滑块后，仿真暂停，模型相应模块随滑块位置变动；手指松开后，仿真继续进行。

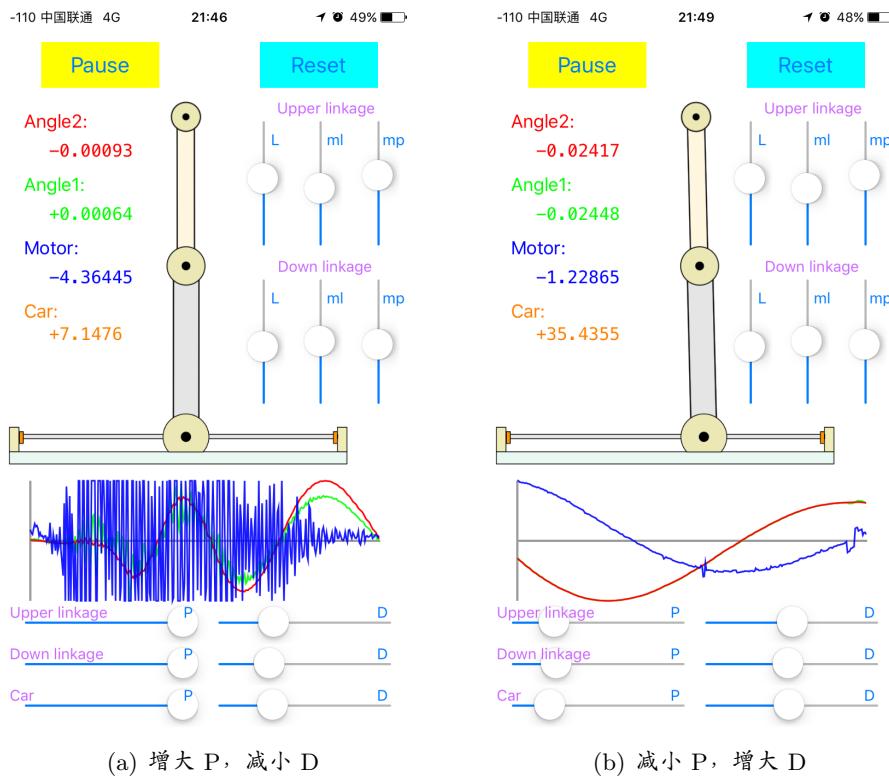


Fig. 3.3: 通过滑块调整控制器参数

3.2.2 调节控制器参数

如3.1中所述，调节下侧六个滑块，可以动态改变控制器PD参数，如图3.3所示。在稍微施加干扰后可见，增大P、减小D参数后系统明显震荡加剧，偏差变小；而减小P，增大D参数后系统更为平稳，但调节速度较慢（示波器最后侧震荡为截屏时的干扰）。

与3.2.1中不同的是，调节PD参数时仿真系统正常运行，并不暂停，因而可以更为迅速和直观地反馈调节效果。

3.2.3 利用重力感应施加干扰

利用智能手机丰富的传感器，尤其是加速度计和陀螺仪模块，可以很好地采集外界对手机施加的扰动信号，作为干扰施加在倒立摆上，较好地模拟真实的噪声信号。

基于 XCode CoreMotion 开发框架，直接提取传感器采集到的手机的倾角及水平方向加速度信号，作为干扰施加在小车上。手机倾斜角信号类似于阶跃信号，而水平方向加速度信号则类似于正负脉冲信号。利用手机可以简单地将这两种信号合成，更真实地测试倒立摆系统的控制性能。

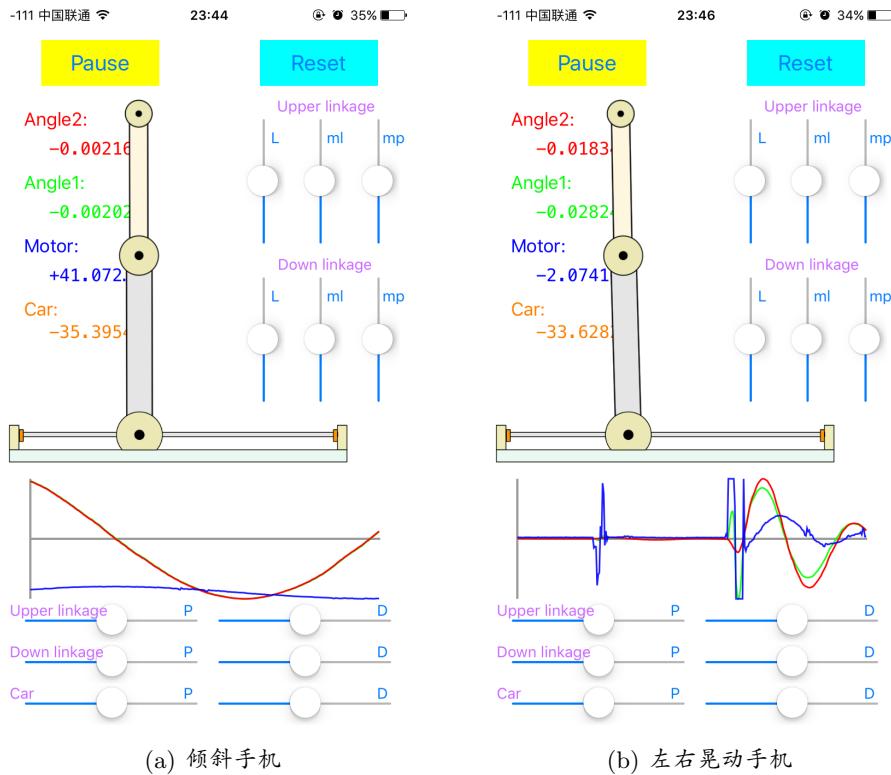


Fig. 3.4: 施加干扰

3.3 实现原理

针对该模型的仿真主要基于式2.5，采用4阶Runge-Kutta方法迭代求解。

考虑该系统为数字控制器对模拟系统进行控制，且由屏幕显示，故需对模拟系统计算步长、数字控制器控制周期、界面刷新频率等分别定义。综合考虑舍入误差、截断误差与系统运行效率，仿真参数取为

- 仿真步长 $h = 5 \text{ ms}$;
- 控制周期 $T_c = 10 \text{ ms}$;
- 干扰信号采集周期 $T_f = 50 \text{ ms}$;
- 界面刷新频率 $f = 48 \text{ FPS}$;

3.3.1 摆数据 linkage 类

在设计时考虑该倒立摆模型是可以调整的，故重心、转动惯量等参数均为变量，需要专门计算和存储，故定义该类。

该类的函数定义如 Code.3.1 所示。其中 calculate() 函数用于计算整个摆（摆杆+摆锤）的重心和绕重心旋转的转动惯量。其余函数则用于模型缩放调整和读取私有变量。

其中，重心计算公式

$$l_i = \frac{\sum_i^n m_i x_{cm,i}}{\sum_i^n m_i} = \frac{x_{cm,l}m_l + x_{cm,p}m_2p}{m_l + m_p} \quad (3.1)$$

细杆绕转轴通过断点与杆垂直的转动惯量取

$$J = \frac{ml^2}{3} \quad (3.2)$$

围绕重心转动的转动惯量可将摆杆等效为两部分，而后对各个部件用平行轴定理[4]

$$J = J_C + md^2 \quad (3.3)$$

合成。

Code 3.1: linkage 类函数

```

1 class linkage {
2     init ()
3     init (Length: Double, LinkWeight: Double, PendulumWeight: Double)
4     private func calculate()
5
6     func zoomLength(times:Float)
7     func zoomWidth(times:Float)
8     func zoomRadius(times:Float)
9
10    func getLength ()->(CGFloat)
11    func getWidth ()->(CGFloat)
12    func getRadius ()->(CGFloat)
13    func getGravityCenter ()->(Double)
14    func getRotaryInertia ()->(Double)
15    func getMass ()->(Double)
16    func getLengthForCalc ()->(Double)
17 }
```

3.3.2 系统仿真 twoOrderInvertedPendulum 类

对上下摆分别建立模型后，需要定义类控制其和小车的运动，故定义此类，如Code.3.2所示。其中：

- simulation() 函数为仿真系统核心函数，采用 4 阶 Runge-Kutta 方法，计算当前帧的小车位置、上下摆偏角，由主控中定时器，根据计算步长 h 触发；
- calculateMatrix() 和 fourOrderRungeKutta() 函数为私有函数，作为仿真计算中的一部分，共同完成式2.5中的计算，由 simulation() 调用；

- `updateModelParameters()` 函数用于在模型（上、下摆）参数改变后，更新式2.5中参数，在模型调整结束后触发；
- `setDisturbance()` 函数用于更新干扰信号，定时采集加速度计等的干扰信号；
- `setControllerParameters()` 函数用于更新控制器参数，在参数滑块对应数值变动时触发，以实现对调节效果的实时观测；
- `updateController()` 函数用于更新控制器输出，在主控中由定时器，根据控制周期 T_c 触发；
- `reset()` 函数用于重置仿真系统，恢复初始（直立）状态，由界面按钮触发；

Code 3.2: twoOrderInvertedPendulum 类函数

```
1 let gravity = 9.8
2 let carMass = 1.0
3 let c0 = 20.0, c1 = 0.0071 * 0.1, c2 = 0.0071 * 0.1
4
5 class twoOrderInvertedPendulum {
6     init(SimulationStep:Double)
7     func updateModelParameters()
8     private func fourOrderRungeKutta()
9     private func calculateMatrix()
10    func setDisturbance(Disturbance:Double?)
11    func setControllerParameters(KpCar1:Float, KdCar1:Float, KpAngle1:
12        Float, KdAngle1:Float, KpAngle2:Float, KdAngle2:Float)
13    func updateController() -> (Double)
14    func reset()
15    func simulation() -> (CGFloat, CGFloat, CGFloat)
16 }
```

3.3.3 示波器曲线 OscilloscopeCurve 类

为方便增加或修改示波器需要显示的曲线，定义该类，如 Code.3.3所示。

共有函数中，`getData()` 函数用于加入新的数据，`updateCurve()` 函数用于刷新示波器曲线，其余共有函数用于设置曲线颜色、位置和绘制坐标轴等。

为绘制仿真系统波形，必须对各个时刻的参数值进行储存，并依次绘制前 N 个点位置。由于 Swift 面向对象语言提供的数组，可以自由实现入队、出队、和遍历等工作，不用建立专门的数据结构，十分方便。

为增加示波器效果，而不增加操作难度，对示波器曲线建立 Auto Scale 功能。即用满刻度表示当前显示波形中绝对值最大的值，其余数值按该尺度缩放。常规做法是每次都寻找需要显示数组中最大值，即遍历数组。为增加计算效率，采用用空间换时间策略，即定义两个私有变量 `dataMax:CGFloat` 和 `dataMaxIndex:Int`，分别用以存储需显示曲线（即当前数组）中绝对值最大元素的值和其索引，并在 `getData()` 函数中，针对每一个即将入队的数

值，进行更新；存储了绝对值最大的数值的位置 dataMaxIndex 后，如后续数据没有绝对值比 dataMax 大的，且在 dataMax 即将移出示波器显示范围时，调用 reAutoScale() 功能，遍历数组，寻找绝对值最大值及其索引。

Code 3.3: OscilloscopeCurve 类函数

```
1 class OscilloscopeCurve: UIView {  
2     func setCurveColor(red: CGFloat, green: CGFloat, blue: CGFloat,  
3                         alpha: CGFloat)  
4     func setPosition(x: Double, y: Double)  
5     func getData(newData: CGFloat)  
6     private func reAutoScale()  
7     func drawAxis(context: CGContextRef)  
8     func updateCurve(context: CGContextRef)  
9 }
```

不足的是，由于作者 iOS 开发经验有限，绘制示波器曲线时，利用各个点创建 path 对象，由 UIView:setNeedsDisplay() 函数更新屏幕。在显示3条曲线，每条曲线320个点时，每次都需要重新绘制 $320 \times 3 = 960$ 个点并连线，更新效率低，占用系统较多资源。

3.3.4 App 主控 ViewController类

作为程序的主控类，该类定义对象和功能函数都较多，如 Code.3.4 所示。

对象主要包括：

- 控制模型对象：invPendulum；
- 用于控制时序的对象：timerSimulation、cmm 等；
- UIImageView 对象，用于显示模型：bottom、car、linkage1、joint1 等；
- UILabel 对象，即数值显示框：labelAngle1 等；
- UISlider 对象，即滑块：kpCar、linkage1LengthSlider 等；

函数主要包括：

- 滑块操作类函数：ControllerParametersChanged() 等；
- 界面初始化函数：drawInit()；
- 定时器设置函数：addTimer()、removeTimer()；
- 系统状态控制函数：systemReset() 等；
- 控制系统仿真函数：simulation()、updateController()；
- 界面刷新函数：refresh()、carRefresh() 等；

- 模型各个部件运动控制函数：carMove() 等；
- App 进程控制函数：viewDidLoad() 等；

Code 3.4: ViewController 类函数

```
1 class ViewController: UIViewController {  
2     @IBAction func ControllerParametersChanged(sender: AnyObject)  
3     @IBAction func resetButtonClicked(sender: AnyObject)  
4     @IBAction func runOrPauseButtonClicked(sender: AnyObject)  
5     @IBAction func modelParametsSliderTouchUp(sender: AnyObject)  
6     @IBAction func modelParametsSliderTouchDown(sender: AnyObject)  
7     @IBAction func linkage1DataChanged(sender: AnyObject)  
8     @IBAction func linkage2DataChanged(sender: AnyObject)  
9  
10    func drawInit()  
11  
12    func addTimer()  
13    func removeTimer()  
14  
15    func systemReset()  
16    func systemRun()  
17    func systemPause()  
18  
19    func simulation()  
20    func updateController()  
21  
22    func refresh()  
23    func carRefresh()  
24    func linkage1Refresh()  
25    func joint1Refresh()  
26    func linkage2Refresh()  
27    func joint2Refresh()  
28  
29    func carMove(position:CGFloat)  
30    func linkage1Rotate(angle:CGFloat)  
31    func linkage2Rotate(angle:CGFloat)  
32  
33    override func viewDidLoad()  
34    override func viewDidAppear(animated: Bool)  
35    override func didReceiveMemoryWarning()  
36    override func viewWillDisappear(animated: Bool)  
37    deinit()  
38 }
```

App 开始时，自动调用 viewDidLoad() 和 viewDidAppear() 函数，主要完成界面初始化 drawInit() 和系统运行 systemRun() 函数。其中 systemRun() 时添加定时器 addTimer() 等。

App 挂起或退出时，自动调用 `viewWillDisappear()` 函数，调用系统暂停 `systemPause()` 函数，移除定时器 `removeTimer()` 等。

考虑模型显示界面运动时，系统采用仿射变换计算新的坐标[5]。当小车位置改变时，必须重新定义与小车相连的下摆的初始坐标，并在此基础上旋转角度；而后上摆根据下摆位置、下摆长度、下摆角度重新定义上摆的初始坐标，并在此基础上旋转相应角度。即 `carMove()` 函数在完成小车移动后，还要调用 `linkage1Refresh()`, `joint1Refresh()`, `linkage2Refresh()`, `joint2Refresh()` 函数，以保证与小车实际相连的部件的正确显示与变换。`linkage1Rotate()` 函数与之相似。

其余功能均通过较为简单的程序编写逻辑实现，在此不再赘述。

3.3.5 数学支持 `mathSupport.swift` 库

考虑模型在仿真过程中需要计算一系列矩阵等数学功能，而 Swift 语言不是专门的数学计算语言，为简化其他程序编写，增加该文件，如 Code.3.5 所示。

Swift 语言中矩阵以数组形式存储，在计算时需指明矩阵的行和列数。同时 Swift 提供了矩阵乘法函数 `vDSP_mmulD()`，但参数过多不易于使用，加法也是一样；同时没有提供矩阵求逆函数。为此，对原有矩阵乘法和加法函数做进一步封装，提升了程序易读性和易用性；考虑实际需求与计算效率，定义了三阶矩阵求逆及负逆矩阵函数。最后，定义了两个内插函数，主要用于电机饱和及角度映射。

Code 3.5: `mathSupport.swift` 库函数

```
1 func square(x:Double)->(Double)
2 func invert3OrderMatrix(source : UnsafePointer<Double>,positiveTarget :
    UnsafeMutablePointer<Double>,negativeTarget : UnsafeMutablePointer<
    Double> )
3 func matrixMultiply(A: UnsafePointer<Double>,B: UnsafePointer<Double>,
    result: UnsafeMutablePointer<Double>,M: vDSP_Length ,N: vDSP_Length ,
    P: vDSP_Length )
4 func matrixAdd(A: UnsafePointer<Double>, B: UnsafePointer<Double>,
    result: UnsafeMutablePointer<Double>,N: vDSP_Length )
5 func insertInPi(var x:Double)->(Double)
6 func insertIn(x:Double ,min:Double ,max:Double)->(Double)
```

Chapter 4

仿真实验

4.1 开环分析

对默认模型参数的系统，将控制器所有 PD 参数归零，观察系统自由摆下，曲线如图4.1所示。其中曲线剧烈震荡，是由于上下摆在左右晃动过程中，角度 θ_i 的值在 $\pm\pi$ 之间震荡。

而实际观察中则明显可以看出，在下落过程中，上摆带动下摆运动，且上摆震荡幅度和频率都明显大于下摆，与真实物理模型一致。

至此，可以说明该系统开环状态与实际系统基本一致，仿真模型建立较为准确。

4.2 闭环分析

4.2.1 LQR 控制器设计

采用 LQR 控制器。针对式2.5定义的系统，在 $\Theta = \mathbf{0}$ 附近线性化。

定义性能指标函数

$$J = \int_0^{\infty} [\mathbf{x}(t)^T \mathbf{Q} \mathbf{x}(t) + u(t)^T \mathbf{R} u(t)] dt \quad (4.1)$$

其中 \mathbf{Q} 定义了各个状态变量误差的权重，为正半定实对称矩阵，为方便选取，通常选定其为对角常阵。对角线上元素 $q_{i,i}$ 越大，则对相应分量 x_i 的误差越重视，分量 x_i 的误差就相对越小。因为系统只有一个输入所以 \mathbf{R} 是一个标量，规定了输入 $u(t)$ 的权值。构建性能指标函数的目标就是确定最优控制规律 $u^*(t)$ 。该式的成立要求系统完全能控。

因为状态方程中

$$\text{rank} [\mathbf{B} \quad \mathbf{AB} \quad \mathbf{A}^2 \mathbf{B} \quad \mathbf{A}^3 \mathbf{B}] = 6 \quad (4.2)$$

故系统完全能控。

$$\text{rank} [\mathbf{C} \quad \mathbf{CA} \quad \mathbf{CA}^2 \quad \mathbf{CA}^3]^T = 6 \quad (4.3)$$

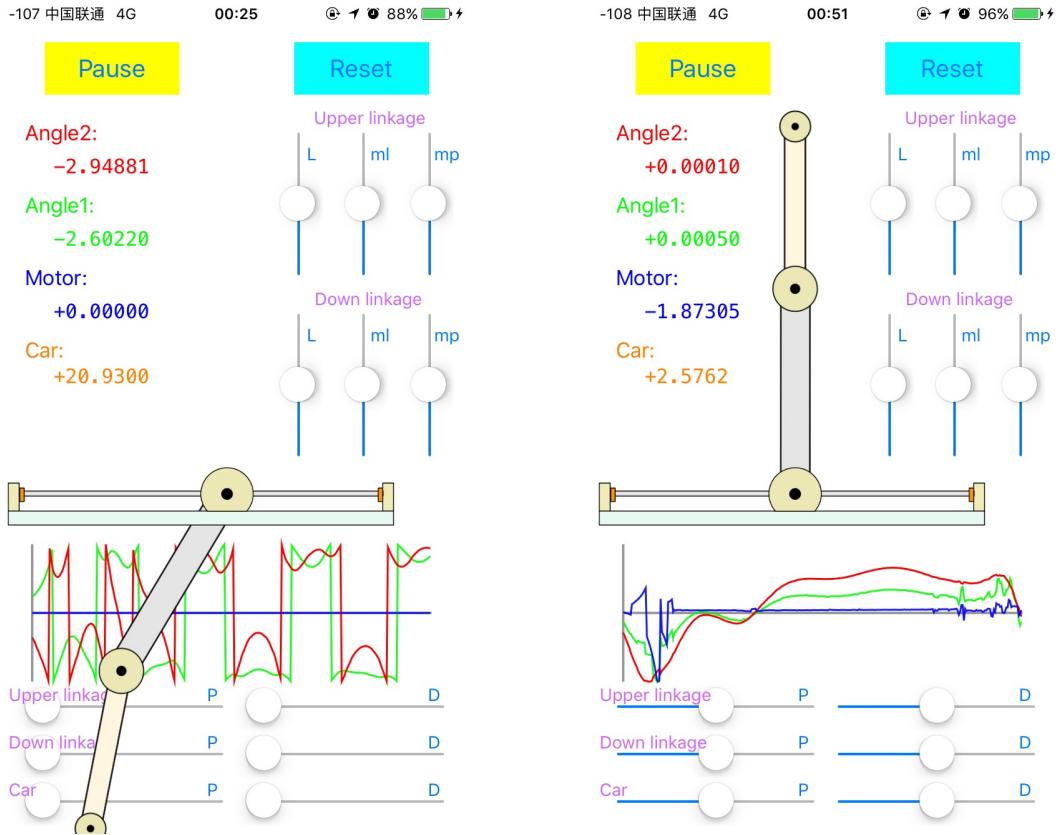


Fig. 4.1: 二阶倒立摆系统开环响应曲线

Fig. 4.2: 二阶倒立摆系统闭环响应曲线

故系统完全能观。

设最优控制规律

$$u^*(t) = -\mathbf{K}\mathbf{x}(t) \quad (4.4)$$

其中, \mathbf{K} 为最优反馈增益。

设 \mathbf{P} 为如下 Riccati 矩阵方程的解

$$-\mathbf{P}\mathbf{A} - \mathbf{A}^T\mathbf{P} + \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P} - \mathbf{Q} = \mathbf{0} \quad (4.5)$$

存在唯一最优控制式

$$u^*(t) = -\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P}\mathbf{x}(t) \quad (4.6)$$

使性能指标取最小值[3]

$$\mathbf{J}^* = \frac{1}{2}\mathbf{x}^{*\top}(t_0)\mathbf{P}\mathbf{x}^*(t_0) \quad (4.7)$$

通过求取合适的反馈增益矩阵, 我们可以使倒立摆系统处于最优控制状态。

由于系统是非线性系统, 因而使用最优二次型整定法的整定结果不一定是误差最小的 \mathbf{K} 。

故采用单纯形法对 K 进行优化。优化指标:误差的绝对值与时间 t 乘积的积分量

$$J = \int t|e|dt \quad (4.8)$$

最终最佳增益矩阵

$$K = [10 \ 479 \ 648 \ 0.814 \ 24.7 \ 95] \quad (4.9)$$

4.2.2 控制器性能分析

系统默认参数时，不是加干扰（自然界存在少量干扰）情况下，响应曲线如图4.2所示。据此可见，增加控制器后系统稳定，且该控制器效果良好。

修改模型参数

详见3.2.1，在此不再赘述。

修改控制器参数

详见3.2.2，在此不再赘述。

施加干扰

详见3.2.3，在此不再赘述。

4.3 误差分析

1. 仿真过程存在舍入误差、截断误差等由于数字化、离散化造成的误差；
2. 没有考虑角度、位置测量准确性与精确度；
3. 电机过于理想，不存在电感等滞后环节；
4. CoreMotion 框架采集到的加速度信号也存在一定误差；
5. 未考虑小车的静摩擦力；

Chapter 5

总结

倒立摆作为十分经典的被控对象，在控制理论领占据着重要的地位。最早接触倒立摆这一模型在哪本书上已经记不清楚了，但第一次真真正正地熟悉倒立摆，对倒立摆做有效控制，还是在 Freescale 智能车上。

上一学年里，我和队友们选择参加了 Freescale 智能车光电平衡组的比赛。从车模的组装到电子电路的连接的硬件层，到底层输入输出驱动和函数接口、简化卡尔曼滤波和互补滤波数字实现的数据层，再到平衡控制、速度控制、方向双闭环控制的控制层，再到最后各个参数的整定与整体优化，每一个任务的完成，每一个层的实现，都凝聚了我们太多的故事汗水。也正是由于这段经历，我们对倒立摆的性能，以及倒立摆的控制，有了十分深入的了解。

平衡环中，为了增进平衡这一最基础控制的性能，我们采用了模糊控制策略，大大提升了车体稳定和抗扰性；速度环中，为避免静差和积分过大对平衡的影响，先后或同时采用遇限削弱积分法、积分分离法、有效偏差法等多种改良式数字 PID 算法，以及反馈-前馈控制策略，使整车速度更加平稳顺畅；方向环中，为消除左右轮阻尼等不平衡因素，采用不同周期的双闭环控制策略，实现了对车体方向精确把控；除此以外，还多次使用惯性滤波，滑动平均滤波，四点中心差分等多种计算机控制技术，进一步提升了车模性能。可谓是运用了从数学分析，到嵌入式开发，再到自动控制、运动控制、过程控制、计算机控制等诸多学科的知识，真正做到了活学活用。最终我们也成功杀入国赛，同全国七十余只参赛队伍，群雄逐鹿。

在有了上述经验后，这学期我又选修了我系赵群飞教授开设的《倒立摆系统课程设计》，从理论上对倒立摆这一系统有了更深入的认识，学习、设计、应用了 LQR 控制器。但是这里的倒立摆和智能车的倒立摆显然不同，因为不能随意地、模拟真实环境地施加干扰。

现阶段智能手机飞速发展，各式各样的传感器极大地丰富了手机的功能，使手机在某些方面能力超过了电脑。为此，在较为了解倒立摆这一系统的基础上，我决心开发一款能真实地模拟二阶倒立摆的手机端 App，用于研究倒立摆这一被控对象，也可用于 PID 调节的教学。

因为以前有一些 iOS 开发经验，所以上手并不是太难。但是以前做的是更为简单的 App，所以这次依然学到了很多 Swift 知识与经验，对 App 框架设计、面向对象编程等都有了更进一步的认识。

编程苦，编程累，编程 bug 全部跪。这次的程序最终有 700+ 行，主体部分是在三个晚上完成的，其后又不断地修修补补，才完成了最后的程序，虽然还有一定的瑕疵和发生概率极低的 bug。但是在 debug 过程中，我更加熟练了断点的相关操作，学会了反差代码的技能。

倒立摆作为经典的被控对象，已经有无数前人学者对其做了深入研究。但在充分利用移动端传感器上的优势，在移动端平台上实现，还是几乎没有的，尤其是最近发布的 Swift 语言。因而本文的创新工作主要在于利用 iOS 平台和 Swift 语言实现，因而在模型推导建立和控制器方面多为借鉴他人成果，在此重申并表示感谢。

世界上有多种多样、形形色色的控制系统，我们不可能逐一对其进行测试和分析，而控制系统仿真为我们的学习和设计提供了极大的帮助。通过这门课的学习，我对控制系统仿真的认识不在停留在 MatLab 语言调用的阶段，更是学会了其仿真实现原理及背后知识，并掌握了利用 MatLab 以外的语言，如 C++、Swift 等仿真实际系统的方法。也充实了在上海交通大学这一所一流高校的最后一年的学习生活。可谓收获满满，不再赘述。

相信这门课一定是我大四上学期里，收获最丰富的一门课程！

Acknowledgements

首先要感谢的是杜秀华老师。杜老师从大三下学期给我们教授《现代控制理论》以来，一直以她清晰、耐心、负责的教学，带我们探索知识的殿堂。这学期班里同学较少，她则更加认真地对待每一个同学，主动关心我们的学习，在解答我们问题的同时，并及时对我们的学习提出指导。这学期因为保研面试、实验室活动等原因，共三次缺席课堂，老师不仅给我准假，也从没有因此责怪我，令我十分感动。

其次感谢我的室友刘泽翔同学。刘泽翔同学不仅在模型建立过程中与我进行了充分的讨论，帮我理清思路，让我及时认清了方案中存在的问题，及时改正少绕弯路，更在我编写程序时的生活作息上给予我极大的理解与照顾，堪称“中国好室友”。

Bibliography

- [1] 黄向华. 控制系统仿真[M]. 北京航空航天大学出版社, 2008.
- [2] 么健石, 侯祥林, 徐心和. 二级倒立摆摆起控制的研究[J]. 控制与决策, 2004, 19(10):1183-1186.
- [3] 施颂椒, 陈学中, 杜秀华. 现代控制理论基础[M]. 高等教育出版社, 2009.
- [4] 上海交通大学物理教研室. 大学物理教程 (上册) [M]. 上海交通大学出版社, 2011.
- [5] Quartz 2D Programming Guide. Apple Inc. https://developer.apple.com/library/ios/documentation/GraphicsImaging/Conceptual/drawingwithquartz2d/dq_affine/dq_affine.html
- [6] Quanser Inc. *QNET Rotpen Workbook (Student)*, 2011.
- [7] 谢剑英, 贾青. 微型计算机控制技术[M]. 国防工业出版社, 2001.
- [8] Golnaraghi F, Kuo B C. *Automatic control systems*[J]. Complex Variables, 2010, 2: 1-1.
- [9] 赵群飞. 倒立摆课程设计, 2015

Appendix A

Swift 语言代码

A. ViewController.swift

```
//  
//  ViewController.swift  
//  invertedPendulum  
//  
//  Created by F on 15/11/24.  
  
//  Copyright © 2015年 F. All rights reserved.  
  
  
import UIKit  
import CoreMotion  
  
var linkage1Data = linkage(Length: 164.66667, LinkWeight: 28,  
PendulumWeight: 60/1.5)  
var linkage2Data = linkage(Length: 194/1.5, LinkWeight: 33/1.5,  
PendulumWeight: 44/1.5)  
//var linkage2Data = linkage(Length: 194/3, LinkWeight: 33/3,  
//PendulumWeight: 44/3)  
//var linkage2Data = linkage(Length: 194/6, LinkWeight: 33/6,  
PendulumWeight: 44/6)  
  
let simulationStep = 0.005  
  
class ViewController: UIViewController {  
  
    let controlPeriod = 0.01  
    let refreshFPS = 48.0  
    let deviceMotionUpdateInterval = 0.05  
  
    let carBoundary = CGFloat(135)
```

```
let bottomWidth = CGFloat(779/2.5),bottomHeight = CGFloat(91/2.5)
let carWidth = CGFloat(110/2.5)
let zeroX = CGFloat(160), zeroY = CGFloat(395)
let oscilloscopeBeginX = 25.0 , oscilloscopeBeginY = 490.0

@IBOutlet weak var labelAngle1: UILabel!
@IBOutlet weak var labelAngle2: UILabel!
@IBOutlet weak var labelMotor: UILabel!
@IBOutlet weak var labelCar: UILabel!

@IBOutlet weak var kpCar: UISlider!
@IBOutlet weak var kdCar: UISlider!
@IBOutlet weak var kpAngle1: UISlider!
@IBOutlet weak var kdAngle1: UISlider!
@IBOutlet weak var kpAngle2: UISlider!
@IBOutlet weak var kdAngle2: UISlider!
@IBAction func ControllerParametersChanged(sender: AnyObject) {
    invPendulum.setControllerParameters(kpCar.value, KdCar1:
    kdCar.value, KpAngle1: kpAngle1.value, KdAngle1: kdAngle1.value,
    KpAngle2: kpAngle2.value, KdAngle2: kdAngle2.value)
//        angle1 = CGFloat( kp1.value * 6.28)
//        angle2 = CGFloat( kd1.value * 6.28)
}

@IBOutlet weak var linkage1LengthSlider: UISlider!
@IBOutlet weak var linkage2LengthSlider: UISlider!
@IBOutlet weak var linkage1widthSlider: UISlider!
@IBOutlet weak var linkage2widthSlider: UISlider!
@IBOutlet weak var linkage1RadiusSlider: UISlider!
@IBOutlet weak var linkage2RadiusSlider: UISlider!
@IBOutlet var myViewTest: myView!
// @IBOutlet weak var labelTest: UILabel!
@IBOutlet weak var runOrPauseButton: UIButton!

@IBAction func resetButtonClicked(sender: AnyObject) {
    systemReset()
    run = false
}
@IBAction func runOrPauseButtonClicked(sender: AnyObject){
    if run{
        systemPause()
    }else{
```

```
        systemRun()
    }
    run = !run
}

@IBAction func modelParametsSliderTouchUp(sender: AnyObject) {
    if run{
        systemRun()
    }
    invPendulum.updateModelParameters()
}
@IBAction func modelParametsSliderTouchDown(sender: AnyObject) {
    systemPause()
}
@IBAction func linkage1DataChanged(sender: AnyObject) {
    linkage1Data.zoomLength(linkage1LengthSlider.value)
    linkage1Data.zoomWidth(linkage1widthSlider.value)
    linkage1Data.zoomRadius(linkage1RadiusSlider.value)
    linkage1Refresh()
    joint1Refresh()
    linkage1Rotate(angle1)

    linkage2Refresh()
    joint2Refresh()
    linkage2Rotate(angle2)
}

@IBAction func linkage2DataChanged(sender: AnyObject) {
    linkage2Data.zoomLength(linkage2LengthSlider.value)
    linkage2Data.zoomWidth(linkage2widthSlider.value)
    linkage2Data.zoomRadius(linkage2RadiusSlider.value)
    linkage2Refresh()
    joint2Refresh()
    linkage2Rotate(angle2)
}

var cmm = CMMotionManager()
let bottom = UIImageView()
let car = UIImageView()
let linkage1 = UIImageView()
let linkage2 = UIImageView()
let joint1 = UIImageView()
let joint2 = UIImageView()
```

```
var invPendulum = twoOrderInvertedPendulum(SimulationStep:  
simulationStep)  
  
var carPosition = CGFloat (0)  
var angle1 = CGFloat (0)  
var angle2 = CGFloat (0)  
var motorOutput = CGFloat (0)  
// var angle1Speed = CGFloat (0.0)  
var accelerationX = CGFloat (0.0)  
var timerSimulation = NSTimer()  
var timerRefresh = NSTimer()  
var timerController = NSTimer()  
var run:Bool = true  
  
func carRefresh(){  
    car.transform = CGAffineTransformIdentity  
    car.frame = CGRectMake(zeroX - carWidth/2, zeroY - carWidth/2,  
carWidth, carWidth)  
}  
func linkage1Refresh(){  
    linkage1.transform = CGAffineTransformIdentity  
    linkage1.frame = CGRectMake(zeroX + carPosition -  
linkage1Data.getWidth()/2 , zeroY -  
linkage1Data.getLength(), linkage1Data.getWidth(), linkage1Data.get  
Length() )  
}  
func joint1Refresh(){  
    joint1.transform = CGAffineTransformIdentity  
    joint1.frame = CGRectMake(zeroX + carPosition -  
linkage1Data.getRadius()/2,  
zeroY-linkage1Data.getLength()-linkage1Data.getRadius()/2 ,  
linkage1Data.getRadius(), linkage1Data.getRadius())  
}  
func linkage2Refresh(){  
    linkage2.transform = CGAffineTransformIdentity  
    linkage2.frame = CGRectMake(joint1.frame.midX -  
linkage2Data.getWidth()/2,  
joint1.frame.midY-linkage2Data.getLength(), linkage2Data.getWidth(  
), linkage2Data.getLength())  
}  
func joint2Refresh(){  
    joint2.transform = CGAffineTransformIdentity  
    joint2.frame =  
CGRectMake(joint1.frame.midX-linkage2Data.getRadius()/2,
```

```
joint1.frame.midY-linkage2Data.getLength()-linkage2Data.getRadius()
)/2 , linkage2Data.getRadius(), linkage2Data.getRadius())
}

func drawInit(){
    linkage1LengthSlider.transform =
CGAffineTransformMakeRotation(CGFloat (-M_PI_2))
    linkage2LengthSlider.transform =
CGAffineTransformMakeRotation(CGFloat (-M_PI_2))
    linkage1widthSlider.transform =
CGAffineTransformMakeRotation(CGFloat (-M_PI_2))
    linkage2widthSlider.transform =
CGAffineTransformMakeRotation(CGFloat (-M_PI_2))
    linkage1RadiusSlider.transform =
CGAffineTransformMakeRotation(CGFloat (-M_PI_2))
    linkage2RadiusSlider.transform =
CGAffineTransformMakeRotation(CGFloat (-M_PI_2))

    myViewTest.oscilloscopeCurveAngle2.setCurveColor(1, green: 0,
blue: 0, alpha: 1)
    myViewTest.oscilloscopeCurveAngle1.setCurveColor(0, green: 1,
blue: 0, alpha: 0.9)
    myViewTest.oscilloscopeCurveMotor.setCurveColor(0, green: 0,
blue: 1, alpha: 0.9)
//    myViewTest.oscilloscopeCurveCarPosition.setCurveColor(1,
green: 0, blue: 1, alpha: 0.9)

myViewTest.oscilloscopeCurveAngle1.setPosition(oscilloscopeBeginX,
y: oscilloscopeBeginY)

myViewTest.oscilloscopeCurveAngle2.setPosition(oscilloscopeBeginX,
y: oscilloscopeBeginY)

myViewTest.oscilloscopeCurveMotor.setPosition(oscilloscopeBeginX,
y: oscilloscopeBeginY)
//
myViewTest.oscilloscopeCurveCarPosition.setPosition(oscilloscopeB
eginX, y: oscilloscopeBeginY)

bottom.image = UIImage(named: "bottom")
bottom.frame = CGRectMake(zeroX-bottomWidth/2 ,
zeroY-bottomHeight/2 - 14 + carWidth/2, bottomWidth, bottomHeight)
```

```
car.image = UIImage(named: "car")
carRefresh()

linkage1.image = UIImage(named: "linkage1")
linkage1Refresh()

joint1.image = UIImage(named: "joint1")
joint1Refresh()

linkage2.image = UIImage(named: "linkage2")
linkage2Refresh()

joint2.image = UIImage(named: "joint2")
joint2Refresh()

self.view.addSubview(linkage1)
self.view.addSubview(linkage2)
self.view.addSubview(bottom)
self.view.addSubview(car)
self.view.addSubview(joint1)
self.view.addSubview(joint2)
}

func simulation(){
    (carPosition,angle1,angle2) = invPendulum.simulation()
}
func updateController(){
    motorOutput = CGFloat( invPendulum.updateController() )
}

func addTimer(){
    if !timerSimulation.valid{
        timerSimulation =
NSTimer.scheduledTimerWithTimeInterval(simulationStep, target:
self, selector: "simulation", userInfo: nil, repeats: true)
    }
    if !timerRefresh.valid{
        timerRefresh =
NSTimer.scheduledTimerWithTimeInterval(1.0 / refreshFPS, target:
self, selector: "refresh", userInfo: nil, repeats: true)
    }
    if !timerController.valid{
```

```
        timerController =
NSTimer.scheduledTimerWithTimeInterval(controlPeriod, target: self,
selector: "updateController", userInfo: nil, repeats: true)
    }
}

func removeTimer(){
    timerSimulation.invalidate()
    timerRefresh.invalidate()
    timerController.invalidate()
}

func carMove(position:CGFloat){
    car.transform = CGAffineTransformIdentity
    car.transform = CGAffineTransformMakeTranslation(position,
0)
    linkage1Refresh()
    joint1Refresh()
    linkage2Refresh()
    joint2Refresh()
}

func linkage1Rotate(angle:CGFloat){
    linkage1.transform = CGAffineTransformIdentity
    linkage1.transform =
CGAffineTransformRotate(CGAffineTransformMakeTranslation(linkage1
Data.getLength()*sin(angle)/2 ,
linkage1Data.getLength()*(1-cos(angle))/2), angle)

    joint1.transform = CGAffineTransformIdentity
    joint1.transform =
CGAffineTransformMakeTranslation(linkage1Data.getLength()*sin(ang
le), linkage1Data.getLength()*(1-cos(angle)))

    linkage2Refresh()
    joint2Refresh()
}

func linkage2Rotate(angle:CGFloat){
    linkage2.transform =
CGAffineTransformRotate(CGAffineTransformMakeTranslation(linkage2
Data.getLength()*sin(angle)/2 ,
linkage2Data.getLength()*(1-cos(angle))/2), angle)
}
```

```
        joint2.transform =
CGAffineTransformMakeTranslation(linkage2Data.getLength()*sin(angle),
linkage2Data.getLength()*(1-cos(angle)))
}

func refresh(){
    labelAngle2.text = String(format: "%+.5f", angle2)
    labelAngle1.text = String(format: "%+.5f", angle1)
    labelMotor.text = String(format: "%+.5f", motorOutput)
    labelCar.text = String(format: "%+.4f", carPosition)
    carMove(carPosition)
    linkage1Rotate(angle1)
    linkage2Rotate(angle2)
    myViewTest.oscilloscopeCurveAngle1.getData(angle1)
    myViewTest.oscilloscopeCurveAngle2.getData(angle2)
    myViewTest.oscilloscopeCurveMotor.getData(motorOutput)
    //
    myViewTest.oscilloscopeCurveCarPosition.getData(carPosition)
        myViewTest.setNeedsDisplay()
        if(abs(carPosition) > carBoundary ){
            invPendulum.reset()
        }
}

func systemReset(){
    systemPause()
    invPendulum.reset()
    carPosition = 0
    angle1 = 0
    angle2 = 0
    refresh()
}

func systemRun(){
    addTimer()
    runOrPauseButton.setTitle("Pause", forState:
    UIControlState.Normal)
    if cmm.deviceMotionAvailable{

cmm.startDeviceMotionUpdatesUsingReferenceFrame(CMAttitudeReferenceFrame.XTrueNorthZVertical, toQueue: NSOperationQueue.mainQueue(),
withHandler: { (data:CMDeviceMotion?, err:NSError?) -> Void in
        if (data != nil){


```

```
self.invPendulum.setDisturbance((data?.userAcceleration.x)! * 50.0
+ (data?.gravity.x)! * 50.0)
    }
}
}

func systemPause(){
removeTimer()
runOrPauseButton.setTitle("Run", forState:
UIControlState.Normal)
if cmm.deviceMotionActive{
cmm.stopDeviceMotionUpdates()
}

}

override func viewDidLoad() {
super.viewDidLoad()
drawInit()
invPendulum.setControllerParameters(kpCar.value, KdCar1:
kdCar.value, KpAngle1: kpAngle1.value, KdAngle1: kdAngle1.value,
KpAngle2: kpAngle2.value, KdAngle2: kdAngle2.value)
}

override func viewDidAppear(animated: Bool) {
cmm.deviceMotionUpdateInterval = deviceMotionUpdateInterval
systemRun()
}

override func didReceiveMemoryWarning() {
super.didReceiveMemoryWarning()
// Dispose of any resources that can be recreated.
}

override func viewWillDisappear(animated: Bool) {
systemPause()
}
deinit{
systemPause()
}
}
```

B. linkage.swift

```
//  
//  linkage.swift  
//  invertedPendulum  
//  
//  Created by F on 15/11/27.  
  
//  Copyright © 2015年 F. All rights reserved.  
  
//  
  
import UIKit  
  
class linkage {  
    private var length: Double = 0  
    private var linkWeight: Double = 0  
    private var pendulumWeight: Double = 0  
    private var lengthZoom: Double = 1  
    private var linkWeightZoom: Double = 1  
    private var pendulumWeightZoom: Double = 1  
    //    var test:  
    //    let A = Mat  
  
    private var l=1.0, ml=1.0, mp=1.0  
    private var gravityCenter = 1.0 , rotaryInertia = 1.0  
  
    init(){}
    init(Length: Double, LinkWeight: Double, PendulumWeight: Double){  
        length = Length  
        linkWeight = LinkWeight  
        pendulumWeight = PendulumWeight  
        calculate()  
    }  
    func calculate(){  
        l = length * lengthZoom / 500  
        ml = linkWeight * linkWeightZoom / 300  
        mp = pendulumWeight * pendulumWeightZoom / 200  
        gravityCenter = (l/2*ml+l*mp)/(ml+mp)  
        rotaryInertia = ml*gravityCenter/l * square(gravityCenter) /  
3  
            + ml*(l-gravityCenter)/l * square(l-gravityCenter) / 3  
            + mp * square(l-gravityCenter)  
    }  
}
```

```
func zoomLength(times:Float){
    lengthZoom = Double(times)
    calculate()
}

func zoomWidth(times:Float){
    linkWeightZoom = Double(times)
    calculate()
}

func zoomRadius(times:Float){
    pendulumWeightZoom = Double(times)
    calculate()
}

func getLength()->(CGFloat){
    return CGFloat(length * lengthZoom)
}

func getWidth()->(CGFloat){
    return CGFloat(linkWeight * linkWeightZoom)
}

func getRadius()->(CGFloat){
    return CGFloat(pendulumWeight * pendulumWeightZoom)
}

func getGravityCenter()->(Double){
    return gravityCenter
}

func getRotaryInertia()->(Double){
    return rotaryInertia
}

func getMass()->(Double){
    return ml + mp
}

func getLengthForCalc()->(Double){
    return l
}
```

C. myView.swift

```
//
// myView.swift
// invertedPendulum
//
// Created by F on 15/11/25.
```



```
// Copyright © 2015年 F. All rights reserved.  
//  
import UIKit  
  
class myView: UIView {  
  
    // Only override drawRect: if you perform custom drawing.  
    // An empty implementation adversely affects performance during  
animation.  
    //    let beginX:CGFloat = 50.0,beginY:CGFloat = 480.0  
  
    var oscilloscopeCurveAngle1 = OscilloscopeCurve()  
    var oscilloscopeCurveAngle2 = OscilloscopeCurve()  
    var oscilloscopeCurveMotor = OscilloscopeCurve()  
//    var oscilloscopeCurveCarPosition = OscilloscopeCurve()  
  
    override func drawRect(rect: CGRect) {  
        let context = UIGraphicsGetCurrentContext()  
        oscilloscopeCurveAngle1.drawAxis(context!)  
        oscilloscopeCurveAngle1.updateCurve(context!)  
        oscilloscopeCurveAngle2.updateCurve(context!)  
        oscilloscopeCurveMotor.updateCurve(context!)  
        oscilloscopeCurveCarPosition.updateCurve(context!)  
    }  
}
```

D. twoOrderInvertedPendulum.swift

```
//  
// twoOrderInvertedPendulum.swift  
// invertedPendulum  
//  
// Created by F on 15/11/25.  
// Copyright © 2015年 F. All rights reserved.  
  
//  
  
import Foundation  
import UIKit
```

```

let gravity = 9.8
let carMass = 1.0
let c0 = 20.0, c1 = 0.0071 * 0.1, c2 = 0.0071 * 0.1

class twoOrderInvertedPendulum {

    private var _angle1:Double = 0.0
    private var _angle2:Double = 0.0
    private var _angle1Speed = 0.0, _angle2Speed = 0.0
    private var _carPosition = 0.0 , _carSpeed = 0.0

    private var _simulationStep:Double = 1
    private var _kpCar=1.0,_kdCar=1.0,_kpAngle1=1.0,_kdAngle1=1.0,
_kpAngle2=1.0,_kdAngle2=1.0
    private var _disturbance = 0.0
    private var drivingForce = 0.0

    private var m0 = carMass, m1=linkage1Data.getMass(),
m2=linkage2Data.getMass(),l1=linkage1Data.getGravityCenter(),L1=l
inkage1Data.getLengthForCalc(),l2=linkage2Data.getGravityCenter()
,J1=linkage1Data.getRotaryInertia(),J2=linkage2Data.getRotaryIner
tia(),g=gravity

    private var MMatrix = [Double](count : 9, repeatedValue : 0.0)
    private var MInvMatrix = [Double](count : 9, repeatedValue : 0.0)
    private var NMInvMatrix = [Double](count : 9, repeatedValue : 0.0)
    private var CMatrix = [c0,0.0, 0,c1+c2,0, 0,0,c2]
    private var GMatrix = [0.0 ,0 ,0]
    private var FMatrix = [0.0, 0, 0]

    private var dotx1 = [0.0, 0, 0],x1=[0.0, 0, 0], dotx2 = [0.0, 0,
0] ,x2=[0.0, 0, 0]

    private var temp33 = [Double](count : 9, repeatedValue : 0.0)
    private var temp31 = [0.0, 0, 0]

    func updateModelParameters(){
        m0 = carMass;m1=linkage1Data.getMass();
m2=linkage2Data.getMass();l1=linkage1Data.getGravityCenter();L1=l
inkage1Data.getLengthForCalc();l2=linkage2Data.getGravityCenter()
    }
}

```

```

;J1=linkage1Data.getRotaryInertia();J2=linkage2Data.getRotaryInertia();
}

private func fourOrderRungeKutta(){
    var x0 = [0.0, 0, 0, 0, 0, 0]
    x0[0] = x1[0]
    x0[1] = x1[1]
    x0[2] = x1[2]
    x0[3] = x2[0]
    x0[4] = x2[1]
    x0[5] = x2[2]
    var k = [[0.0, 0, 0, 0, 0, 0],[0.0, 0, 0, 0, 0, 0],[0.0, 0,
0, 0, 0, 0],[0.0, 0, 0, 0, 0, 0]]

    dif()
    k[0][0] = dotx1[0]
    k[0][1] = dotx1[1]
    k[0][2] = dotx1[2]
    k[0][3] = dotx2[0]
    k[0][4] = dotx2[1]
    k[0][5] = dotx2[2]

    for i in 0..<3{      x1[i] = x0[i] + k[0][i] * _simulationStep
/ 2 }
    for i in 3..<6{      x2[i-3] = x0[i] + k[0][i] * _simulationStep
/ 2 }

    dif()
    k[1][0] = dotx1[0]
    k[1][1] = dotx1[1]
    k[1][2] = dotx1[2]
    k[1][3] = dotx2[0]
    k[1][4] = dotx2[1]
    k[1][5] = dotx2[2]

    for i in 0..<3{
        x1[i] = x0[i] + k[1][i] * _simulationStep / 2
    }
    for i in 3..<6{
        x2[i-3] = x0[i] + k[1][i] * _simulationStep / 2
    }
    dif()
    k[2][0] = dotx1[0]
}

```



```
k[2][1] = dotx1[1]
k[2][2] = dotx1[2]
k[2][3] = dotx2[0]
k[2][4] = dotx2[1]
k[2][5] = dotx2[2]

for i in 0..<3{
    x1[i] = x0[i] + k[2][i] * _simulationStep
}
for i in 3..<6{
    x2[i-3] = x0[i] + k[2][i] * _simulationStep
}
dif()
k[3][0] = dotx1[0]
k[3][1] = dotx1[1]
k[3][2] = dotx1[2]
k[3][3] = dotx2[0]
k[3][4] = dotx2[1]
k[3][5] = dotx2[2]

for i in 0..<3{
    x1[i] = x0[i] +
(k[0][i]+2*k[1][i]+2*k[2][i]+k[3][i])*_simulationStep/6;
}
for i in 3..<6{
    x2[i-3] = x0[i] +
(k[0][i]+2*k[1][i]+2*k[2][i]+k[3][i])*_simulationStep/6;
}

}

private func dif(){
    calculateMatrix()
    dotx1[0] = x2[0]
    dotx1[1] = x2[1]
    dotx1[2] = x2[2]
    matrixMultiply(NMInvMatrix, B: CMatrix, result: &temp33, M: 3,
N: 3, P: 3)
    matrixMultiply(temp33, B: x2, result: &dotx2, M: 3, N: 3, P:
1)
    matrixMultiply(NMInvMatrix, B: GMatrix, result: &temp31, M: 3,
N: 3, P: 1)
    dotx2[0] += temp31[0]
    dotx2[1] += temp31[1]
    dotx2[2] += temp31[2]
```



```
matrixMultiply(MInvMatrix, B: FMatrix, result: &temp31, M: 3,
N: 3, P: 1)
    dotx2[0] += temp31[0]
    dotx2[1] += temp31[1]
    dotx2[2] += temp31[2]

}

private func EulerMethod(){
    dif()
    for i in 0..<3{
        x1[i] += dotx1[i] * _simulationStep
        x2[i] += dotx2[i] * _simulationStep
    }
}

private func calculateMatrix(){
    MMatrix[0] = m0+m1+m2
    MMatrix[1] = (m1*l1 + m2*L1) * cos(x1[1])
    MMatrix[2] = m2*l2*cos(x1[2])
    MMatrix[3] = MMatrix[1]
    MMatrix[4] = J1+m1*square(l1)+m2*square(L1)
    MMatrix[5] = m2*L1*l2*cos(x1[1]-x1[2])
    MMatrix[6] = MMatrix[2]
    MMatrix[7] = MMatrix[5]
    MMatrix[8] = J2+m2*square(l2)
    invert3orderMatrix(MMatrix, positiveTarget: &MInvMatrix,
negativeTarget: &NMInvMatrix)

    //      CMatrix[0] = c0
    CMatrix[1] = -(m1*l1+m2*L1)*sin(x1[1])*x2[1]
    CMatrix[2] = -m2*l2*sin(x1[2])*x2[2]
    //      CMatrix[3] = 0
    //      CMatrix[4] = c1+c2
    CMatrix[5] = m2*L1*l2*sin(x1[1]-x1[2])*x2[2]-c2
    //      CMatrix[6] = 0
    CMatrix[7] = -m2*L1*l2*sin(x1[1]-x1[2])*x2[1]-c2
    //      CMatrix[8] = c2

    GMatrix[1] = -(m1*l1+m2*L1)*g*sin(x1[1])
    GMatrix[2] = -m2*g*l2*sin(x1[2])
    FMatrix[0] = _disturbance + drivingForce
}
```

```

init(SimulationStep:Double){
    _simulationStep = SimulationStep
}
func setDisturbance(Disturbance:Double?){
    _disturbance = Disturbance!
}
func setControllerParameters(KpCar1:Float,
KdCar1:Float,KpAngle1:Float, KdAngle1:Float,KpAngle2:Float,
KdAngle2:Float){
    _kpCar = Double( KpCar1 )
    _kdCar = Double( KdCar1 )
    _kpAngle1 = Double( KpAngle1 )
    _kdAngle1 = Double( KdAngle1 )
    _kpAngle2 = Double( KpAngle2 )
    _kdAngle2 = Double( KdAngle2 )
}
func updateController()->(Double){
    if ((abs(_angle1) > M_PI_4) || (abs(_angle2) > M_PI_4)){
//        drivingForce = -(_carPosition*100*_kpCar +
100*_carSpeed*_kdCar)
        drivingForce = 0
    }else{
        drivingForce = -(_carPosition*1*_kpCar -
479*_angle1*_kpAngle1 + 648*_angle2*_kpAngle2
+ 0.814*_carSpeed*_kdCar - 24.7*_angle1Speed*_kdAngle1
+ 95*_angle2Speed*_kdAngle2 )
    }
    drivingForce = insertIn(drivingForce,min: -50.0,max: 50.0)
    return drivingForce
}
func reset(){
    _carPosition = 0
    _carSpeed = 0
    _angle1 = 0
    _angle1Speed = 0
    _angle2 = 0
    _angle2Speed = 0

    dotx1 = [0.0, 0, 0]
    x1=[0.0, 0, 0]
    dotx2 = [0.0, 0, 0]
}

```



```
x2=[0.0, 0, 0]
}

func simulation() -> (CGFloat, CGFloat, CGFloat) {
    // EulerMethod()
    fourOrderRungeKutta()
    _carPosition = x1[0]
    _angle1 = insertInPi(x1[1])
    _angle2 = insertInPi(x1[2])
    _carSpeed = x2[0]
    _angle1Speed = x2[1]
    _angle2Speed = x2[2]

    return (CGFloat(_carPosition * 100), CGFloat(_angle1), CGFloat(_angle2))
}
```

E. OscilloscopeCurve.swift

```
//  
// OscilloscopeCurve.swift  
// invertedPendulum  
//  
// Created by F on 15/11/28.  
// Copyright © 2015年 F. All rights reserved.  
//  
  
import UIKit  
  
class OscilloscopeCurve: UIView {  
  
    private let dataLengthMax = 320  
    private let dataYScale:CGFloat = 55  
  
    private var beginXx:CGFloat = 50.0,beginYy:CGFloat = 480.0  
    private var data:Array<CGFloat> = [0.0 ,0.0]  
    private var dataMax:CGFloat = 0.0001 ,dataMaxIndex = 1  
    private var dataQueueFull = false
```

```
private var color = Color()

func setCurveColor(red: CGFloat, green: CGFloat, blue: CGFloat,
alpha: CGFloat){
    color.red = red
    color.green = green
    color.blue = blue
    color.alphe = alpha
}

func setPosition(x:Double,y:Double){
    beginXx = CGFloat(x)
    beginYy = CGFloat(y)
}

func getData(newData:CGFloat){
    data.append(newData)
    if dataQueueFull{
        data.removeFirst()
        dataMaxIndex--
    }else if data.count == dataLengthMax{
        dataQueueFull = true
    }

    if abs(newData)>=dataMax{
        dataMax = abs(newData)
        if dataQueueFull{
            dataMaxIndex = dataLengthMax
        }else{
            dataMaxIndex = data.count
        }
    }
    if dataMaxIndex==0{
        reAutoSize()
    }
}

private func reAutoSize(){
    dataMax = 0.0001
    for i in 1 ..< dataLengthMax {
        if abs(data[i]) >= dataMax{
```

```
        dataMaxIndex = i
        dataMax = abs(data[i])
    }
}
}

func drawAxis(context:CGContextRef){
    CGContextMoveToPoint(context, beginXx, beginYy)
    CGContextAddLineToPoint(context, beginXx +
    CGFloat( dataLengthMax), beginYy)

    CGContextMoveToPoint(context, beginXx, beginYy-dataYScale)
    CGContextAddLineToPoint(context, beginXx,
beginYy+dataYScale)

    CGContextSetRGBStrokeColor(context, 0, 0, 0, 0.4)
    CGContextSetLineWidth(context, 2)
    CGContextStrokePath(context)
}

func updateCurve(context:CGContextRef ){
    let path = CGPathCreateMutable()
    CGPathMoveToPoint(path, nil, beginXx,
beginYy+dataYScale*data.first!/dataMax)
    if dataQueueFull{
        for i in 1 ..< dataLengthMax {
            CGPathAddLineToPoint(path, nil, beginXx+CGFloat(i),
beginYy+dataYScale*data[i]/dataMax)
        }
    }else{
        for i in 1 ..< data.count {
            CGPathAddLineToPoint(path, nil, beginXx+CGFloat(i),
beginYy+dataYScale*data[i]/dataMax)
        }
    }
    CGContextSetLineWidth(context, 1.5)
    CGContextSetRGBStrokeColor(context, color.red, color.green,
color.blue, color.alphe)
    CGContextAddPath(context, path)
    CGContextStrokePath(context)
}

struct Color {
```

```

var red = CGFloat(0)
var green = CGFloat(0)
var blue = CGFloat(0)
var alphe = CGFloat(0)
init(){}
}

```

F. mathSupport.swift

```

// 
//  mathSupport.swift
//  invertedPendulum
//
//  Created by F on 15/11/26.
//  Copyright © 2015年 F. All rights reserved.
//

import Accelerate

func square(x:Double)->(Double){
    return x * x
}

func
invert30rderMatrix(source:UnsafePointer<Double>,positiveT
arget:UnsafeMutablePointer<Double>,negativeTarget:UnsafeM
utablePointer<Double> ){
    let
a=source[0],b=source[1],c=source[2],d=source[3],e=source[4]
,f=source[5],g=source[6],h=source[7],i=source[8]
    let det:Double = a*e*i - a*f*h - b*d*i + b*f*g + c*d*h -
c*e*g
    positiveTarget[0] = (e*i - f*h) / det
    positiveTarget[1] = (c*h - b*i) / det
    positiveTarget[2] = (b*f - c*e) / det
    positiveTarget[3] = (f*g - d*i) / det
    positiveTarget[4] = (a*i - c*g) / det
    positiveTarget[5] = (c*d - a*f) / det
    positiveTarget[6] = (d*h - e*g) / det
    positiveTarget[7] = (b*g - a*h) / det
    positiveTarget[8] = (a*e - b*d) / det
    for i in 0..<9{
        negativeTarget[i] = -positiveTarget[i]
    }
}

```

}

```
func matrixMultiply(A: UnsafePointer<Double>,B:  
UnsafePointer<Double>,result:  
UnsafeMutablePointer<Double>,M: vDSP_Length,N:  
vDSP_Length,P: vDSP_Length){  
    vDSP_mmulD(A, 1, B, 1, result, 1, M, P, N)  
}  
  
func matrixAdd(A: UnsafePointer<Double>, B:  
UnsafePointer<Double>, result:  
UnsafeMutablePointer<Double>,N: vDSP_Length){  
    vDSP_vaddD(A, 1, B, 1, result, 1,N )  
}  
//func insertInPi(x:Double)->(Double){  
//    if x > M_PI{  
//        return x - 2*M_PI  
//    }  
//    else if x < -M_PI{  
//        return x + 2*M_PI  
//    }  
//    return x  
//}  
func insertInPi(var x:Double)->(Double){  
    while x > M_PI{  
        x -= 2 * M_PI  
    }  
    while x < -M_PI{  
        x += 2 * M_PI  
    }  
    return x  
}  
  
func insertIn(x:Double,min:Double,max:Double)->(Double){  
    if x<min{  
        return min  
    }  
    else if x>max{  
        return max  
    }  
    return x  
}
```