Chapter 5

Simulations - Pin

5.1 Pin-0.2 2D frictionless circles

5.1.1 Summary

Pin-0.2 is a 2-dimensional frictionless collision simulation for circular bodies. It is geared toward a system of 1 ball and 10 pins, but is extensible to n bodies. The mechanics stem from the first 2 chapters of [7] and are summarized in the following equations ...

5.1.2 Impact equations

The relative velocity of B and B' is defined as:

$$\mathbf{v} \equiv \mathbf{V} - \mathbf{V}' \tag{5.1}$$

The velocities of B and B', as well as their relative velocity, increment as follows:

$$\mathbf{V} = \mathbf{V}_0 + \frac{\mathbf{p}}{M} \tag{5.2}$$

$$\mathbf{V}' = \mathbf{V}'_0 + \frac{\mathbf{p}}{M'} \tag{5.3}$$

$$\mathbf{v} = \mathbf{v}_0 + \frac{\mathbf{p}}{m} \tag{5.4}$$

where m is the effective mass

$$m = \frac{MM'}{M+M'} \tag{5.5}$$

The compressive impulse p_c is the amount of impulse required to halt compression. In other words, it is the impulse needed to bring the normal component of relative velocity to 0.

$$p_c = -v_3 m \tag{5.6}$$

The energetic coefficient of restitution e_* is defined in terms of the work done in the normal direction during compression and restitution:

$$e_*^2 \equiv -\frac{W_n(p_f) - W_n(p_c)}{W_n(p_c)}$$
(5.7)

With some effort, it follows that

$$p_f = p_c(1+e_*) (5.8)$$

$$= -mv_0(1+e_*) \tag{5.9}$$

where v_0 refers to the initial value of the normal component of relative velocity. This can be substituted into equations 5.2 and 5.3:

$$v_0 = (\mathbf{V}_0 - \mathbf{V}'_0) \cdot \hat{\mathbf{n}}_3$$
 (5.10)

$$\mathbf{V}_f = \mathbf{V}_0 - \frac{mv_0}{M} (1 + e_*) \hat{\mathbf{n}}_3 \tag{5.11}$$

$$\mathbf{V}_{f}' = \mathbf{V}_{0}' + \frac{mv_{0}}{M'} (1 + e_{*})\hat{\mathbf{n}}_{3}$$
(5.12)

where ν_0 is the normal component of the incident velocity, V_0 and V'_0 are the initial velocities of the 2 bodies, M and M' are the masses, and m is the effective mass. Note that these equations must be performed in a coordinate system spanned by vectors tangent and normal to the both bodies at the contact point.

5.1.3 Collision detection

Contact is detected by comparing the distance between the centers of 2 objects to the sum of their radii:

$$\sqrt{(x-x')^2 + (y-y')^2} \Leftrightarrow r + r' \tag{5.13}$$

This needs to be checked for every pair of objects.

5.1.4 Implementation

These equations are computed for each collision. The state vector includes 4 quantities per body:

- x speed
- y speed
- x position
- y position

ode113 is used to compute the constant speed trajectories of the bodies and event location is used to determine the any pair of bodies begin contact.

The code is organized into subfunctions, subsubfuctions, and subsubsubfunctions as shown in Figure 5.1.

Collision detection is implemented in an events function:

```
% ...
for i = 2:nBodies
    for j = 1:i-1
        value( (i-1)*nBodies + j ) = sqrt(dx^2 + dy^2) - r(i) - r(j);
    end
end
% ...
```

These i and j indices avoid redundant checks (such as i, j = 1, 2 followed by i, j = 2, 1). The values are packed into a vector only because Matlab does not appear to accept them in matrix form. The indices of the bodies i and j must be deciphered by another function later:

```
i = floor(ie/nBodies) + 1;
j = ie - (i-1)*nBodies;
```



Figure 5.1: Function call graph for Pin-0.2.5 simulation.



Figure 5.2: Pre-collision



5.2 Pin-0.3 2D circles with friction

Pin-0.3 is a 2-dimensional collision simulation for circular bodies with friction. It is geared toward a system of 1 ball and 10 pins, but is extensible to n bodies. The mechanics stem from Chapter 3 of [7]. The code organization is identical to that of Pin-0.2 (see Figure 5.1). 2 major changes have been made to the code to incorporate friction during collisions.

5.2.1 Expanded state vector

The state vector consists of 6n elements:

$$\mathbf{velocity} = \begin{bmatrix} \dot{x}_1 \\ \dot{y}_1 \\ \vdots \\ \dot{x}_n \\ \dot{y}_n \end{bmatrix}$$
(5.14)
$$\mathbf{position} = \begin{bmatrix} x_1 \\ y_1 \\ \vdots \\ x_n \\ y_n \end{bmatrix}$$
(5.15)
$$\mathbf{angularvelocity} = \begin{bmatrix} \dot{\theta}_1 \\ \vdots \\ \dot{\theta}_n \end{bmatrix}$$
(5.16)
$$\mathbf{angle} = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$
(5.17)
$$\mathbf{y} = \begin{bmatrix} \mathbf{velocity} \\ \mathbf{position} \\ \mathbf{angle} \end{bmatrix}$$
(5.18)

5.2.2 Collision handling

Knowns

After transforming the velocities of the 2 colliding bodies into a coordinate system with \hat{n}_1 along the common normal and \hat{n}_2 along the common tangent, the following quantities are known:

Table 5.1: Notation and meaning of known quantities prior to impact

V_0	initial velocity of C
V_0'	initial velocity of C'
\check{M}	mass of B
M'	mass of B'
e_*	energetic coefficient of restitution

And the following parameters are computed from knowns:

$$m = \frac{MM'}{M+M'} \tag{5.19}$$

$$\beta_1 = 1 + \frac{mr_3^2}{M\hat{k}_r^2} + \frac{mr_3'^2}{M'\hat{k}_r'^2}$$
(5.20)

$$\beta_2 = \frac{mr_1r_3}{M\hat{k}_r^2} + \frac{mr_1'r_3'}{M'\hat{k}_r'^2}$$
(5.21)

$$\beta_3 = 1 + \frac{mr_1^2}{M\hat{k}_r^2} + \frac{mr_1'^2}{M'\hat{k}_r'^2}$$
(5.22)

For collinear impacts, $\beta_2 = 1$ and $\beta_3 = 0$.

Intermediate calculations

$$\mathbf{v}(0) = (\hat{\mathbf{V}}_0 + \boldsymbol{\omega}_0 \times \mathbf{r}) - (\hat{\mathbf{V}}'_0 + \boldsymbol{\omega}'_0 \times \mathbf{r}')$$
(5.23)

$$\hat{s} = \operatorname{sign}(v_1(0)) \tag{5.24}$$

Impulse at peak compression for collinear impact:

$$p_c = \frac{-v_3(0)m}{\mu\beta_2\hat{s} + \beta_3} \tag{5.25}$$

$$= -v_3(0)m$$
 (5.26)

Impulse at onset of stick:

$$p_s = \frac{v_1(0)m}{\mu\beta_1\hat{s} + \beta_2}$$
(5.27)

Impulse at separation for collinear impact:

$$p_f^* = p_c(1 + e_*) \tag{5.28}$$

Final impulse:

$$\mathbf{p_f} = \begin{cases} \begin{bmatrix} -\hat{s}\mu p_f^* & 0 & p_f^* \\ -\hat{s}\mu p_s & 0 & p_f^* \end{bmatrix} & p_s > p_f^* \\ p_s < p_f^* \end{bmatrix}$$
(5.29)

In the case $p_s < p_f^*$, p_s is used in the tangential component because no impulse is imparted after stick in the former direction of slip.

Results

The final impulse is used to compute the post-collision velocities and angular velocities:

$$\hat{\mathbf{V}}_{p_f} = \hat{\mathbf{V}}_0 + \frac{\mathbf{p_f}}{M} \tag{5.30}$$

$$\hat{\mathbf{V}}_{p_f}' = \hat{\mathbf{V}}_0' + \frac{\mathbf{p}_f'}{M'} \tag{5.31}$$

$$\boldsymbol{\omega}_{p_f} = \boldsymbol{\omega}_0 + \frac{\mathbf{r} \times \mathbf{p_f}}{M \hat{k}_r^2} \tag{5.32}$$

$$\boldsymbol{\omega}_{p_f}' = \boldsymbol{\omega}_0' + \frac{\mathbf{r}' \times \mathbf{p}_f'}{M' \hat{k}_r'^2}$$
(5.33)

where $\mathbf{p}_{\mathbf{f}}' = -\mathbf{p}_{\mathbf{f}}$ and $\mathbf{r}' = -\mathbf{r}$.

5.2.3 Collision handling code

```
function [y2] = postimpactv(y,ie,e,m)
% i is prime in Stronge's notation
% j is unprime
% using 1 normal, 2 tangential (left), and 3 up
% in contrast to Stronge's 3 normal, 1 tangential (right), 2 down
global r kr
format compact
mu = 0.1;
[i j] = findij(ie,y)
n = size(y, 2);
meff = m(i)*m(j)/(m(i)+m(j));
vi = [v(2*i-1) v(2*i) 0]
vj = [y(2*j-1) y(2*j) 0]
wi = [0 \ 0 \ y(n*4/6+i)]
wj = [0 \ 0 \ y(n*4/6+j)]
ri = [r(i) 0 0]
rj = [-r(j) \ 0 \ 0]
vci = vi + cross(wi,ri)
vcj = vj + cross(wj,rj)
v0 = vcj - vci
shat = sign(v0(2))
beta(1) = 1 + meff*r(j)^2/m(j)/kr(j)^2 + meff*r(i)^2/m(i)/kr(i)^2
beta(2) = 0; % for collinear collisions
beta(3) = 1; % for collinear collisions
% pc = -v0(1)*meff/(mu*beta(2)*shat+beta(3))
pc = -v0(1) * meff
ps = -v0(2)*meff/(mu*beta(1)*shat+beta(2))
pfstar = pc*(1+e)
if ps>pfstar
    pf = [pfstar shat*mu*pfstar 0]
else
    pf = [pfstar shat*mu*ps 0]
end
```

```
vi2 = vi - 1/m(i)*pf
vj2 = vj + 1/m(j)*pf
wi2 = wi - 1/m(i)/kr(i)^2 * cross(ri,pf)
wj2 = wj + 1/m(j)/kr(j)^2 * cross(rj,pf)
KEi0 = 1/2*m(i)*dot(vi,vi) + 1/2*m(i)*kr(i)^2*dot(wi,wi)
KEj0 = 1/2*m(j)*dot(vj,vj) + 1/2*m(j)*kr(j)^2*dot(wj,wj)
KEif = 1/2*m(i)*dot(vi2,vi2) + 1/2*m(i)*kr(i)^2*dot(wi2,wi2)
KEjf = 1/2*m(j)*dot(vj2,vj2) + 1/2*m(j)*kr(j)^2*dot(wj2,wj2)
dKE = KEjf + KEif - KEjO - KEiO
if dKE>0, error('~~~-- Kinetic energy increased!? ---~~'), end
y^{2} = y;
y2([2*i-1 2*i]) = vi2(1:2);
y2([2*j-1 2*j]) = vj2(1:2);
y2(n*4/6+i)
              = wi2(3);
             = wj2(3);
y2(n*4/6+j)
format loose
```

5.2.4 Pinfall by position and entry angle experiment

64 simulations were run to test the relationship between pinfall and position and entry angle from $y = 0.35, 0.375, 0.4, \ldots, 0.525$ m from the right gutter and $\psi = 0, 1, 2, \ldots, 7^{\circ}$. Pinfall was totalled as the number of pins that deviated from their initial positions. The pins unmoved were also noted.

All the data



Figure 5.4: Pins felled by position and entry angle. 0.3574 m is glancing the right side of the 1 pin, 0.4425 m is "flush", and 0.5271 m is the middle of the 1 pin. Each square represents 1 data point.

Pins by position or by angle

Future improvements

- Use a 3D model
- Increase the resolution of angles and positions
- Refine the definition of position For this experiment initial position was defined as [l - r_{ball} - r_{pin} y]. It may be better to couple the x and y positions to account for the curvature of the ball and pins. As stated



Figure 5.5: Pins felled by position.

Figure 5.6: Pins felled by entry angle.

before, y = 0.3574m is a glancing blow and y = 0.5271m is a head-on impact, but these y values correspond to different x values (l and $l - r_{ball} - r_{pin}$, respectively.)

5.3 Pin-0.4 3D frictionless spheres

5.4 Pin-0.6 2 identical spheres

Simulates the collision of 2 identical spheres. The spheres can have any initial speed, position, and angular velocity.

State vector is organized / structured as follows:

```
function x = unpackstate(state)
x = [state.x state.y state.z ...
state.xdot state.ydot state.zdot ...
state.wx state.wy state.wz];
```

5.4.1 Collision processing

xNew

5.4.2

```
function xNew = collision(x)
theta = findtheta(x)
xTheta = transformcoordinates(x, 2,theta);
phi = findphi(xTheta)
xThetaPhi = transformcoordinates(xTheta,1,phi);
xNewThetaPhi = computevelocity(xThetaPhi);
xNewTheta = transformcoordinates(xNewThetaPhi,1,-phi);
```

= transformcoordinates(xNewTheta,

2,-theta);

Velocity computation

```
function xNew = computevelocity(x)
global M R KR MU ESTAR
state = packstate(x);
% 1 is prime
% 2 is unprime
VHat
           = [state.xdot(2) state.ydot(2) state.zdot(2)]
VHatPrime = [state.xdot(1) state.ydot(1) state.zdot(1)]
           = [state.wx(2)]
                             state.wy(2)
omega
                                           state.wz(2)]
omegaPrime = [state.wx(1)
                             state.wy(1)
                                           state.wz(1)]
           = [0 \ 0 \ -R]
r
           = [0 \ 0 \ R]
rPrime
V
       = VHat
                   + cross(omega, r)
VPrime = VHatPrime + cross(omegaPrime, rPrime)
       = V - VPrime
v
alpha = 2/M + R^2 / (M*KR^2);
```

```
= diag([alpha alpha 1])
mInv
pc = -M/2*v(3)
s = sqrt(v(1)^2 + v(2)^2)
ps = s / (MU * mInv(1,1))
pf = pc * (1 + ESTAR)
phi = atan(v(2) / v(1))
if isnan(phi), phi = 0, end
       = [-MU*cos(phi)*min(ps,pf) -MU*sin(phi)*min(ps,pf) pf]
р
pPrime = -p
VHatFinal
                = VHat
                            + p/M
VHatPrimeFinal = VHatPrime - p/M
omegaFinal
                = omega
                           + cross(r,p)
omegaPrimeFinal = omegaPrime + cross(rPrime,pPrime)
% Form post-collision state
newstate = state;
newstate.xdot = [VHatPrimeFinal(1) VHatFinal(1)];
newstate.ydot = [VHatPrimeFinal(2) VHatFinal(2)];
newstate.zdot = [VHatPrimeFinal(3) VHatFinal(3)];
newstate.wx = [omegaPrimeFinal(1) omegaFinal(1)];
              = [omegaPrimeFinal(2) omegaFinal(2)];
newstate.wy
              = [omegaPrimeFinal(3) omegaFinal(3)];
newstate.wz
xNew = unpackstate(newstate);
% Kinetic energy
KEinitial = 1/2*M*norm(VHat)^2
                                     + ...
    1/2*M*norm(VHatPrime)^2
                                     + ...
    1/2*M*KR<sup>2</sup>*norm(omega)<sup>2</sup>
                                     + ...
    1/2*M*KR^2*norm(omegaPrime)^2
KEfinal = 1/2*M*norm(VHatFinal)^2 + ...
    1/2*M*norm(VHatPrimeFinal)^2
                                    + ...
    1/2*M*KR^2*norm(omegaFinal)^2
                                    + ...
```

1/2*M*KR^2*norm(omegaPrimeFinal)^2