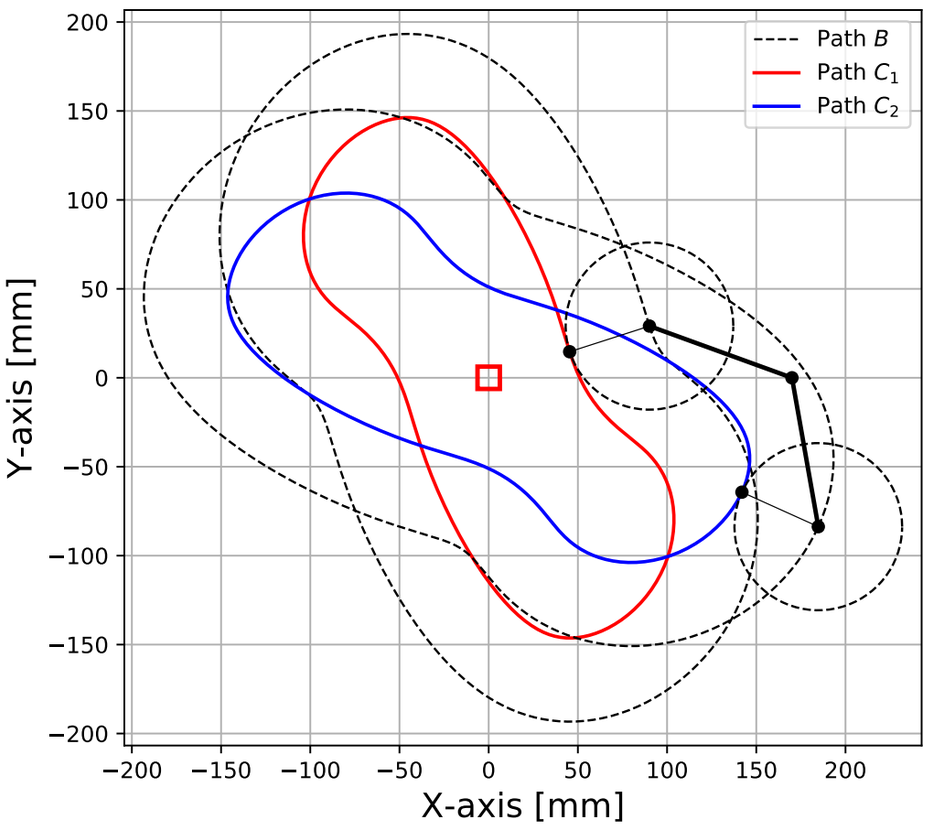
Geometry optimisation for the Marchetti cams



By Wesley Peijnenburg

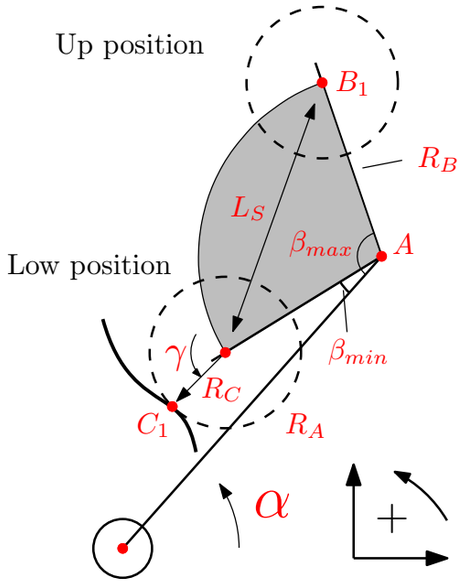
Updated 19 March 2020

Introduction

This article builds further upon two articles already published on marchetti-engine.com:

1. ”Estimating historical dimensions of the Marchetti Engine - December 2016”
2. ”Mathematical solution of the Marchetti cams - March 2020”

The goal of this article to provide the tools necessary to optimise the geometry of the cams. The first interesting aspect to optimise is rotational acceleration of the wheels and to minimise it by changing input parameters. The parameter definitions and solutions are taken from the article ”Mathematical solution of the Marchetti cams” and repeated shortly below:



Parameter	Value	Description
A	-	Rocker pivoting point
B	-	Wheel centre point
C	-	Contact point wheel and cam
R_A	170mm	Radius center to pivoting point
R_B	85mm	Rocker arm length
R_C	47mm	Wheel radius
L_S	-	Cylinder stroke
α	-	Angle to point A; main axis orientation
β_{max}	100°	Maximum angle of rocker arm
β_{min}	20°	Minimum angle of rocker arm
$\Delta\beta$	-	Difference β_{max} and β_{min}
γ	-	Angle from B_1 to contact point C_1

Figure 1: Parameter definition

The equation of the cam geometry:

$$C_{1x} = B_{1x} - R_C \frac{B'_{1y}}{\sqrt{B'^2_{1x} + B'^2_{1y}}}$$
$$C_{1y} = B_{1y} + R_C \frac{B'_{1x}}{\sqrt{B'^2_{1x} + B'^2_{1y}}}$$

Where the curves B are defined by the following equations:

$$B_{1x} = A_x + R_B \cos(\alpha + \pi - \beta_1)$$
$$B_{1y} = A_y + R_B \sin(\alpha + \pi - \beta_1)$$

And their derivatives:

$$B'_{1x} = -R_A \sin(\alpha) - R_B \sin\Big(\alpha + \pi - \beta_1\Big)\Big(1 - \Delta\beta \sin(2\alpha)\Big)$$
$$B'_{1y} = R_A \cos(\alpha) + R_B \cos\Big(\alpha + \pi - \beta_1\Big)\Big(1 - \Delta\beta \sin(2\alpha)\Big)$$

With the swinging motion defined by the angle equations:

$$\beta_1 = \beta_{min} + \frac{1}{2}\Delta\beta\,(1 - \cos(2\alpha))$$

Rotational acceleration of the wheels

Knowing acceleration of the wheels is useful, because acceleration can contribute to slip (and therefore wear) due to wheel inertia. It can be reduced by tweaking parameters, typically at the cost of cylinder stroke distance. Before deriving the solution, two examples are used to explain the method used to calculate it. The first example is a wheel rotating inside a circular hole that is slightly larger than the wheel. The second example is a wheel rotating around a point.

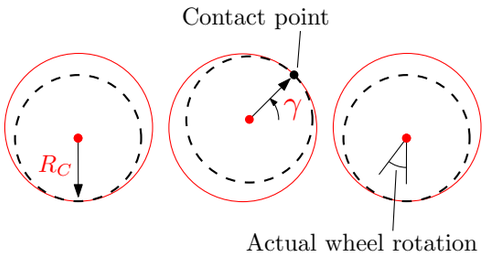


Figure 2: Example 1, wheel in a hole

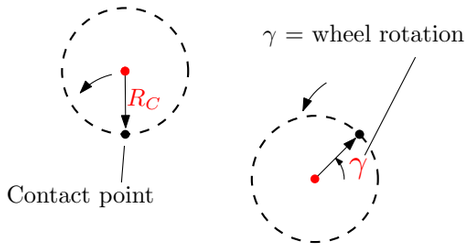


Figure 3: Example 2, wheel around a point

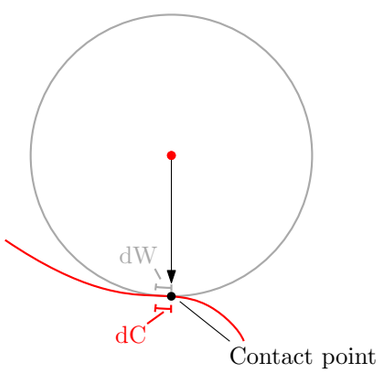
In example 1, the wheel rotation is equal to the distance travelled by the wheel (projected on circumference), minus the circumferential distance travelled by the contact point. Since they are almost equal, the wheel has not rotated much. In example two, the wheel moves around a point and therefore travels no distance. The wheel angle then is equal to the contact point angle γ. For a straight path where the contact point angle γ is constant, wheel rotation is proportional to the distance travelled.

In order to proceed and obtain both γ and the distance travelled along the curve C, numerical methods are applied. Firstly gamma is obtained:

$$\gamma = \tan^{-1}\left(\frac{B'_{1x}}{-B'_{1y}}\right)$$

Where the arc tangent is defined in the algorithm such that it can output an angle starting at 0° up to and excluding 360°. The positive direction is counter clockwise. The X and Y terms are switched because the slope angle B'_1 is rotated 90° inwards, so $X_\gamma = -B'_{1y}$ and $Y_\gamma = B'_{1x}$.

The following additional parameters are defined:



Parameter	Description
$d\alpha$	Differential angle
dw	Differential distance on wheel
dC	Differential distance on cam contour
dS	Differential distance of wheel
sW	Wheel distance (sum total of dS)
vW	Wheel velocity
aW	Wheel acceleration
θ	Wheel revolutions (rad)
$\dot{\theta}$	Wheel rotational velocity ($\frac{rad}{rad_\alpha}$)
$\ddot{\theta}$	Wheel rotational acceleration ($\frac{rad}{rad_\alpha^2}$)

Figure 4: Parameters for wheel rotation

The angle α is discretised in N steps such that each angular step is defined as:

$$d\alpha = \alpha_{n+1} - \alpha_n$$

Then the differential distances are as follows:

$$dw = R_C\left(\gamma_{n+1} - \gamma_n\right)$$
$$dC = \sqrt{(C_{x,n+1} - C_{x,n})^2 + (C_{y,n+1} - C_{y,n})^2}$$

Example 1 in Figure 2, is used here to show that the contact point moves in opposite direction to the actual wheel rotation. Therefore in order to obtain the actual wheel rotation defined here as a circumferential distance, the two variables are added and not subtracted:

$$dS = dw + dC$$

Circumferential distance (proportional to wheel rotations) is defined as the integral over dS:

$$sW_n = \sum_{n=1}^{n} dw_n + dC_n$$

Then, the circumferential velocity and acceleration are obtained:

$$vW_n = \frac{dS}{d\alpha}$$
$$aW_n = \frac{vW}{d\alpha}$$

The acceleration aW_n is plotted in Figure 5 below:

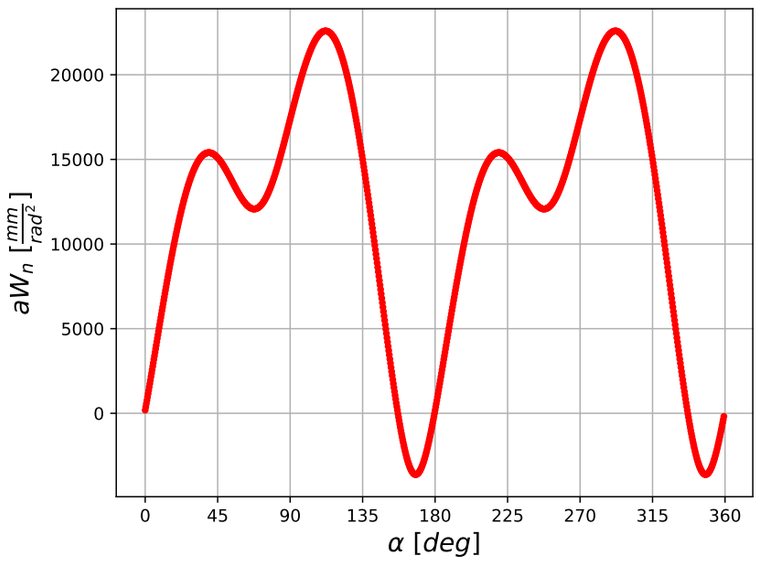


Figure 5: Circumferential acceleration on the wheel

Finally the rotations, rotational velocity and acceleration are as follows:

$$\theta_n = \frac{sW_n}{R_c}, \qquad \dot{\theta}_n = \frac{vW_n}{R_c}, \qquad \ddot{\theta}_n = \frac{aW_n}{R_c}$$

Here it can be assumed that the engine rotates at for example one $\frac{rad_\alpha}{s}$. Then the units of $\dot{\theta}_n$, $\ddot{\theta}_n$ become $\frac{rad}{s}$ and $\frac{rad}{s^2}$ respectively.

Note of the author

As addendum to this article, I have included my Python code to make it easier for others to use this work for their own projects or contribute.

On a personal note, I prefer an analytical solution to a numerical one but have not been able to solve the line integral $\int C(x, y)\, dS$ where $dS = \sqrt{dx^2 + dy^2}\, d\alpha$. Help from a fine mathematician is requested here to advance the search for an analytical solution to this problem (and/or correct the current solution). He or she shall be duly credited and can always contact me at wpeijnenburg@gmail.com.

Appendix - Python code example

```
import sys
import numpy as np
from math import sqrt
from matplotlib import pyplot as plt
from matplotlib.animation import FuncAnimation as func_ani

# Shortened function notations:
def d2r(deg) : return np.deg2rad(deg)
def r2d(rad) : return np.rad2deg(rad)
def sin(rad) : return np.sin(rad)
def cos(rad) : return np.cos(rad)
def atan(x,y): # Redefined to span over 0−360 degrees
    if y>=0   : return np.arctan2(y,x)
    else      : return np.arctan2(y,x) + 2*np.pi

#  User input:
Ra         = 170 # mm, radial distance to point A, the rotating points
Rb         = 85  # Length of the arms
Rc         = 47  # Radius of wheels
beta_min = d2r(20)  # Lowest  swing angle of the arms
beta_max = d2r(100) # Highest swing angle of the arms
res        = 1000 # Resolution
lims       = 0.55*Ra # Boundaries for plotting

# Calculated input:
pi         = np.pi
beta_D    = beta_max − beta_min
alphas    = np.linspace(0, 2*pi, res, endpoint=False) # Main axis orientation
stroke    = 2 * Rb * sin(beta_D / 2) # Stroke distance of the pistons

# Obtaining the geometry
def Ax      (a): return Ra * cos(a)
def Ay      (a): return Ra * sin(a)
def beta1   (a): return beta_min + 0.5*beta_D * (1 − cos(2*a))
def beta2   (a): return beta_min + 0.5*beta_D * (1 + cos(2*a))
def B1x     (a): return Ax(a) + Rb * cos(a + pi − beta1(a))
def B1y     (a): return Ay(a) + Rb * sin(a + pi − beta1(a))
def B2x     (a): return Ax(a) + Rb * cos(a + pi + beta2(a))
def B2y     (a): return Ay(a) + Rb * sin(a + pi + beta2(a))
def dB1x    (a): return −Ra*sin(a) − Rb*sin(a + pi − beta1(a))*(1 − beta_D * sin(2*a))
def dB1y    (a): return  Ra*cos(a) + Rb*cos(a + pi − beta1(a))*(1 − beta_D * sin(2*a))
def dB2x    (a): return −Ra*sin(a) − Rb*sin(a + pi + beta2(a))*(1 − beta_D * sin(2*a))
def dB2y    (a): return  Ra*cos(a) + Rb*cos(a + pi + beta2(a))*(1 − beta_D * sin(2*a))
def C1x     (a): return B1x(a) − Rc * dB1y(a) / sqrt(dB1x(a)**2 + dB1y(a)**2)
def C1y     (a): return B1y(a) + Rc * dB1x(a) / sqrt(dB1x(a)**2 + dB1y(a)**2)
def C2x     (a): return B2x(a) − Rc * dB2y(a) / sqrt(dB2x(a)**2 + dB2y(a)**2)
def C2y     (a): return B2y(a) + Rc * dB2x(a) / sqrt(dB2x(a)**2 + dB2y(a)**2)
def gamma1 (a): return atan(−dB1y(a), dB1x(a))
def wheel1x(a, rot): return [B1x(a) + Rc * cos(t+rot) for t in alphas] # Includes rotation angle for plotting
def wheel1y(a, rot): return [B1y(a) + Rc * sin(t+rot) for t in alphas] # Includes rotation angle for plotting
def wheel2x(a): return [B2x(a) + Rc * cos(t) for t in alphas]
def wheel2y(a): return [B2y(a) + Rc * sin(t) for t in alphas]

# Evaluation of cam distance travelled, wheel orientation, velocity and acceleration
ao             = 0 # Alpha old
Wdiff_prev     = Rc * ( gamma1(ao) − gamma1(alphas[−1]−2*pi)) # Diff. circumferential distance on wheel
Cdiff_prev     = sqrt( (C1x(ao)−C1x(alphas[−1]))**2 + (C1y(ao)−C1y(alphas[−1]))**2 ) # diff. on cam
Wrotdist_prev= Cdiff_prev + Wdiff_prev # Diff. distance of wheel
Vw_old         = Wrotdist_prev / (ao − alphas[−1] + 2*pi) # circumferential velocity of wheel
Lcw            = [0]        # Wheel circumferential distance travelled
Vcw            = [Vw_old] # Wheel circumferential velocity
Acw            = []         # Wheel circumferential acceleration

for i, an in enumerate(alphas[1:]):
    da             = an−ao # Delta alpha (step size)

    # Detect jump of 360 degrees to 0 and correct step size
    if abs(gamma1(an) − gamma1(ao)) > pi:
        Wdiff      = Rc * ( gamma1(an) − gamma1(ao) + 2*pi)
    else:
        Wdiff      = Rc * ( gamma1(an) − gamma1(ao))

    Cdiff          = sqrt( (C1x(an)−C1x(ao))**2 + (C1y(an)−C1y(ao))**2 )
    Wrotdist_new   = Cdiff + Wdiff # Circumferential distance of wheel
    Vw_new         = Wrotdist_new / da # Circumferential velocity between two points
    Lcw.append( Lcw[i] + Wrotdist_new )  # Sum and store total for each alpha
```

```
    Vcw.append( Vw_new )                    # Sum and store velocity
    Acw.append( (Vw_new - Vw_old) / da ) # Sum and store acceleration

    #Update variables
    ao              = an
    Vw_new          = Vw_old
    Wrotdist_prev = Wrotdist_new

W1_orients       = [L/Rc + gamma1(0)  for L in Lcw] # Radians

# Animation of the solution
fig          = plt.figure();
ax           = plt.axes(xlim=(-lims, lims), ylim=(-lims, lims)); ax.grid();ax.axis('equal')
plt.title('Marchetti engine, by Wesley Peijnenburg')
ax.set_xlabel('X [mm]', fontsize=15); ax.set_ylabel('Y [mm]', fontsize=15)
plotA,      = ax.plot([], [], 'ro' , linewidth=2, markersize=7)
plotAs,     = ax.plot([], [], 'k--', linewidth=1, markersize=7)
plot0,      = ax.plot([], [], 'k+' , markeredgewidth=2, markersize=15)
plotB1,     = ax.plot([], [], 'r-o')
plotB2,     = ax.plot([], [], 'b-o')
plotB1s,    = ax.plot([], [], 'r--', linewidth=0.5)
plotB2s,    = ax.plot([], [], 'b--', linewidth=0.5)
plotC1s,    = ax.plot([], [], 'r--')
plotC2s,    = ax.plot([], [], 'b--')
plotW1,     = ax.plot([], [], 'k:' , linewidth=2)    # Wheels
plotW2,     = ax.plot([], [], 'k-' , linewidth=0.5) # Wheels
plotW1C,    = ax.plot([], [], 'm-' , linewidth=0.5) # Contact points
plotW2C,    = ax.plot([], [], 'm-' , linewidth=0.5) # Contact points
plotROTC1,= ax.plot([], [], 'g-' , linewidth=3)    # Wheel actual rotations

def init(): # initialization function: plot the background of each frame
    # Static:
    plot0   .set_data([0], [0])
    plotAs .set_data([Ax (a) for a in alphas], [Ay (a) for a in alphas])
    plotB1s.set_data([B1x(a) for a in alphas], [B1y(a) for a in alphas])
    plotB2s.set_data([B2x(a) for a in alphas], [B2y(a) for a in alphas])
    plotC1s.set_data([C1x(a) for a in alphas], [C1y(a) for a in alphas])
    plotC2s.set_data([C2x(a) for a in alphas], [C2y(a) for a in alphas])

    # Dynamic:
    plotA      .set_data([], [])
    plotB1     .set_data([], [])
    plotB2     .set_data([], [])
    plotW1     .set_data([], [])
    plotW2     .set_data([], [])
    plotW1C    .set_data([], [])
    plotW2C    .set_data([], [])
    plotROTC1.set_data([], [])

    return plotA,


def animate(i): # animation function, called sequentially
    a = d2r(i/res * 360)
    plotA   .set_data( Ax(a), Ay(a) )
    plotB1 .set_data([Ax(a), B1x(a)], [Ay(a), B1y(a)])
    plotB2 .set_data([Ax(a), B2x(a)], [Ay(a), B2y(a)])
    plotW1 .set_data(wheel1x(a, W1_orients[i]), wheel1y(a, W1_orients[i]))
    plotW2 .set_data(wheel2x(a), wheel2y(a))
    plotROTC1.set_data([B1x(a), B1x(a)+cos(W1_orients[i])*Rc],\
                       [B1y(a), B1y(a)+sin(W1_orients[i])*Rc])
    plotW1C.set_data([B1x(a), C1x(a)], [B1y(a), C1y(a)])
    plotW2C.set_data([B2x(a), C2x(a)], [B2y(a), C2y(a)])

    return plotA, plotB1, plotB2, plotW1, plotW2, plotW1C, plotW2C, plotROTC1

anim = func_ani(fig, animate, init_func=init, frames=res, interval=15, blit=True)
plt.show()
```