Warsaw University of Technology

FACULTY OF
POWER AND AERONAUTICAL ENGINEERING

Institute of Heat Engineering

Half-semester Project

for subject "Computer Methods in Combustion"

Simulating HCCI engine in Python Cantera Package

Hubert Demianowski
Michał Forysiak
Aleksander Kipiela

supervisor
dr inż. Mateusz Żbikowski

Warsaw, 2019

Abstract

This paper was written as a half-semester project for “Computer Methods in Combustion”, subject conducted on 6th semester of Aerospace Engineering, specialization Propulsion Systems, in Warsaw University of Technology, faculty of Power and Aeronautical Engineering. Its main purpose was to show students possibilities of Cantera Python. For authors, it was first project executed in this environment, so it is mostly based on examples accessible on Cantera official website.

Keywords: Python, Cantera, Piston engine, HCCI

Contents

1 Introduction

1.1 Conventional Engines

Conventional solution of combustion, like spark-ignition in petrol engines and compression-ignition in Diesel engines, are well known in nowaday world. First of it - spark controlled - is an internal combustion engine, where the combustion process of the air-fuel mixture is ignited by a spark from spark plug. This is contrast to second type of mentioned engine, compression-ignition, typically Diesel engines, where the heat generated from compression together with the injection of fuel is enough to initiate the combustion process, without needing any external spark.

1.2 HCCI Engines

Homogeneous charge compression ignition (HCCI) combines characteristics of conventional gasoline engine and diesel engines. This type of ignition is a form of internal combustion in which fuel and oxidizer are compressed to the point of auto-ignition. HCCI injects fuel during the intake stroke. However, rather than using an electric discharge to ignite a portion of the mixture, HCCI raises density and temperature by compression until the entire mixture reacts spontaneously.

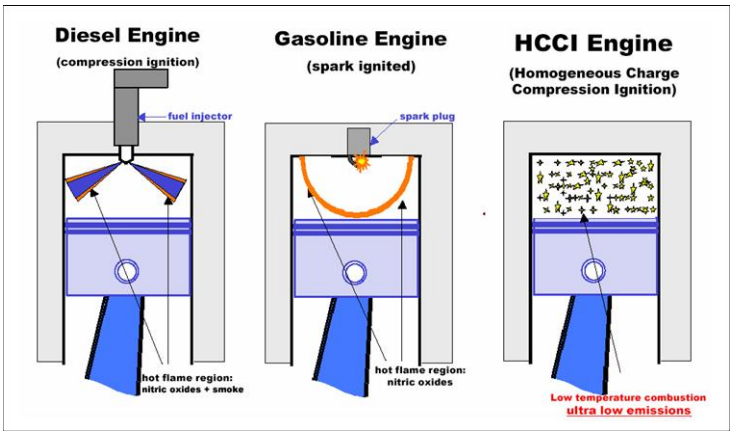

Figure 1: Comparison of Diesel, Gasoline & HCCI engines
$Source : https://www.wired.com/2007/08/gms-hcci-engine/$

Main advantages of HCCI engines are:

- Homogeneous mixing of fuel and air leads to cleaner combustion and lower emissions. Because peak temperatures are significantly lower than in typical SI engines, NOx levels are almost negligible. Additionally, the technique does not produces soot.
- HCCI engines can theoretically operate on gasoline, diesel fuel and most alternative fuels.
- HCCI engines can operate at diesel-like compression ratios (>15), thus achieving 30% higher efficiencies than conventional SI gasoline engines.

Main disadvantages:

- Autoignition is difficult to control, unlike the ignition event in SI and diesel engines, which are controlled by spark plugs and in-cylinder fuel injectors respectively.
- High heat release and pressure rise rates contribute to faster engine wear.

– HCCI engines have a small power range, constrained at low loads by lean flammability limits and high loads by in-cylinder pressure restrictions.

HCCI is more difficult to control than other combustion engines, such as SI and diesel. In a typical gasoline engine, a spark is used to ignite the pre-mixed fuel and air. In Diesel engines, combustion begins when the fuel is injected into pre-compressed air. In both cases, combustion timing is explicitly controlled. In an HCCI engine, however, the homogeneous mixture of fuel and air is compressed and combustion begins whenever sufficient pressure and temperature are reached. This means that no well-defined combustion initiator provides direct control.

Mazda however came with a resolution for this problem. In their newly released engine, HCCI is controlled by increase of pressure, caused by spark plug ignition of local richer air-fuel mixture, which is near the spark plug. Local richer air-fuel mixture is provided by timed fuel injector, which injects fuel directly near the spark plug.

1.3 Aim of project

Main goal of this project is to analyze theoretical possibilities of using methane as fuel in HCCI engine. To examine this we used Cantera and investigated input and output parameters, their values and theoretical possibilities combined with practical abilities of engine engineering technique.

2 Model Description

Engine was modeled as Cantera zero-dimensional reactors network. This means that properties of gas are the same in whole volume of cylinder, thus it is not possible for example to model flame propagation. Inlet, outlet and ambient air were modeled as Cantera reservoirs. Inlet valve was modeled as Cantera valve connecting inlet reservoir and cylinder, modeled as Cantera reactor. Outlet valve connects cylinder and outlet reservoir. Piston was modeled as Cantera wall, set between reactor in cylinder and ambient air. Everything is marked on scheme below.

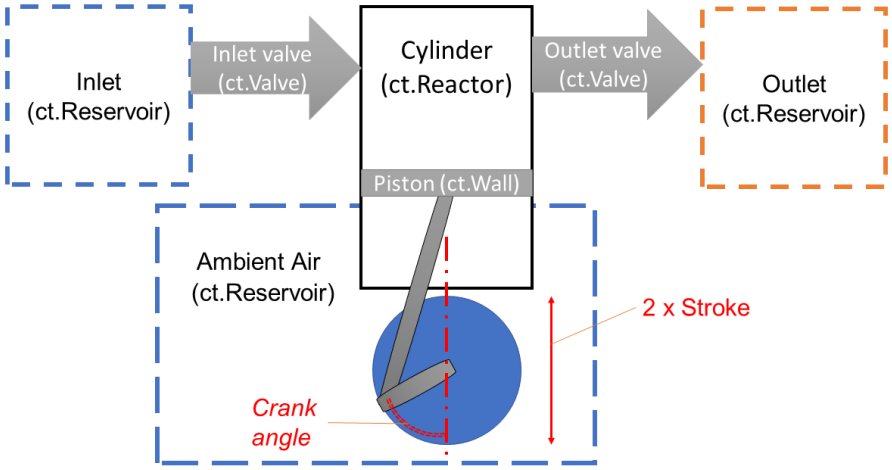

Figure 2: Modeled engine scheme

Hydrogen in stoichiometric mix with air was used as a fuel. Simulation was conducted for 3 rotational engine speeds: 1500, 2500 and 3500 rpm. All data (for example geometry) can be derived from python code (posted at the end of this document).

3 Results

As a result of calculations, following plots were obtained:

3.1 1500 RPM

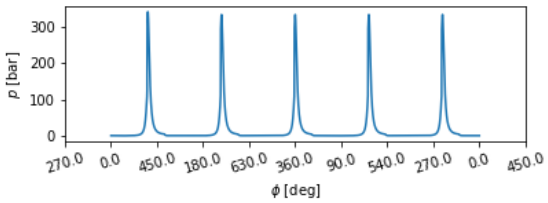

Figure 3: P(phi) 1500 RPM

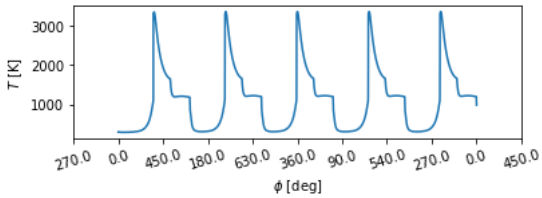

Figure 4: T(phi) 1500 RPM

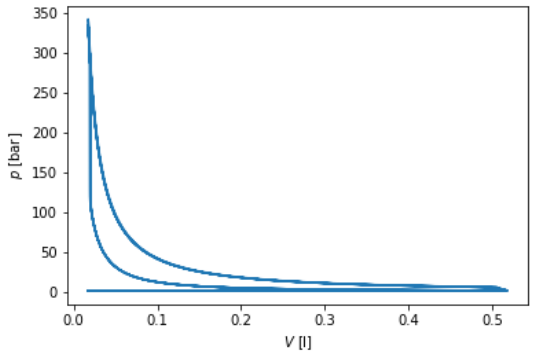

Figure 5: P(V) 1500 RPM

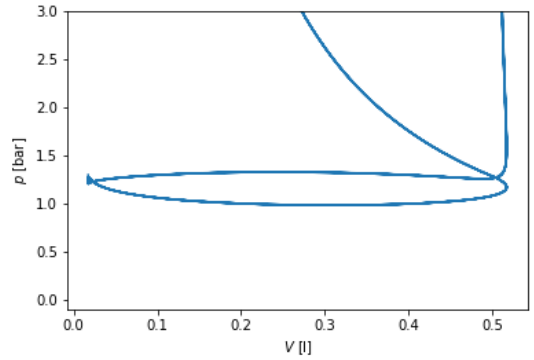

Figure 6: P(V) inlet loop 1500 RPM

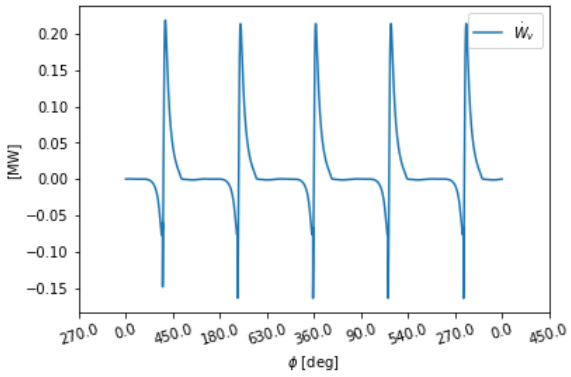

Figure 7: Work 1500 RPM

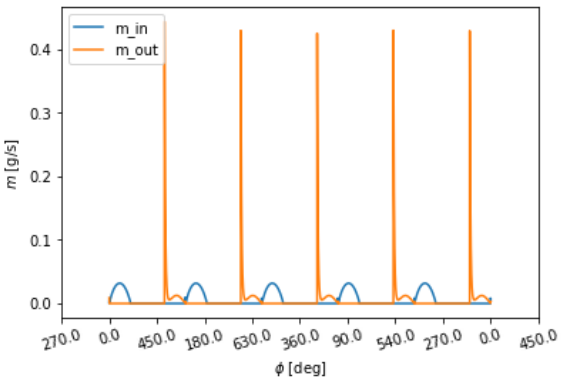

Figure 8: Mass flow 1500 RPM

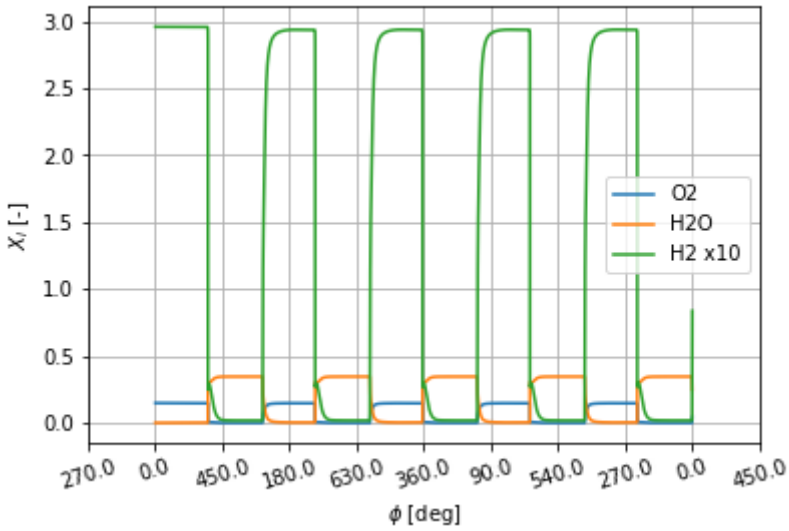

Figure 9: Gas composition 1500 RPM

3.2 2500 RPM

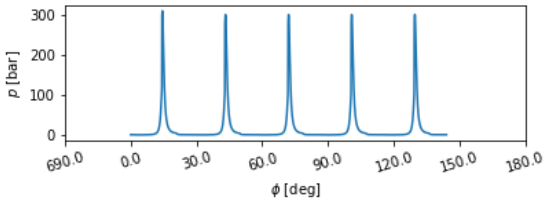

Figure 10: P(phi) 2500 RPM

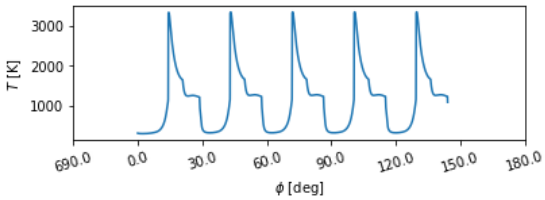

Figure 11: T(phi) 2500 RPM

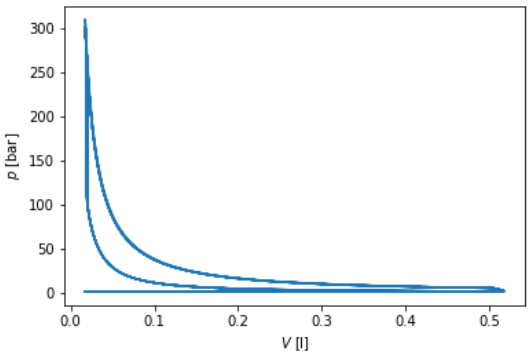

Figure 12: P(V) 2500 RPM

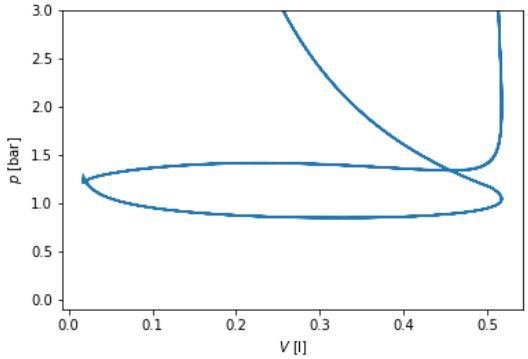

Figure 13: P(V) inlet loop 2500 RPM

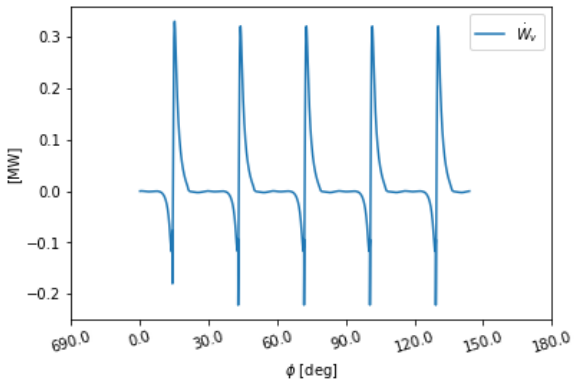

Figure 14: Work 2500 RPM

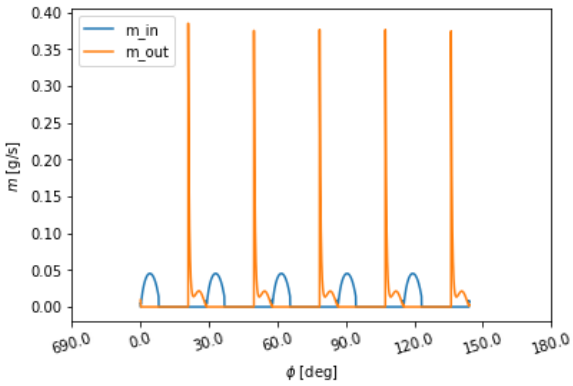

Figure 15: Mass flow 2500 RPM

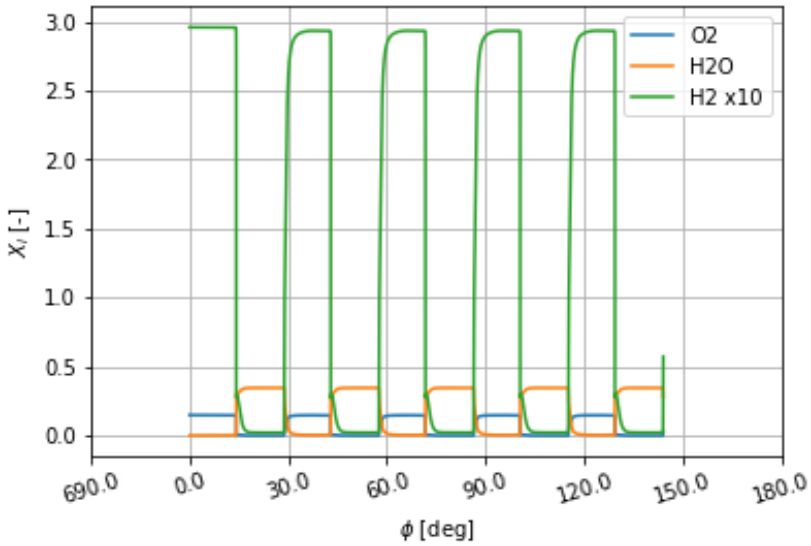

Figure 16: Gas composition 2500 RPM

3.3 3500 RPM

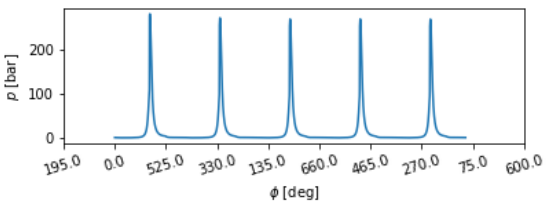

Figure 17: P(phi) 3500 RPM

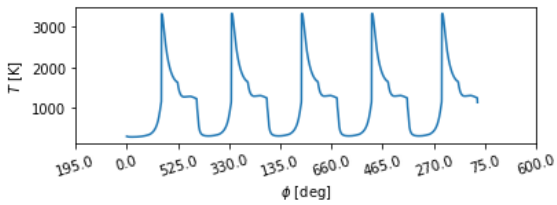

Figure 18: T(phi) 3500 RPM

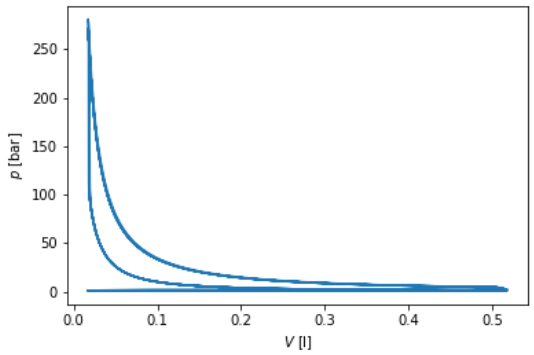

Figure 19: P(V) 3500 RPM

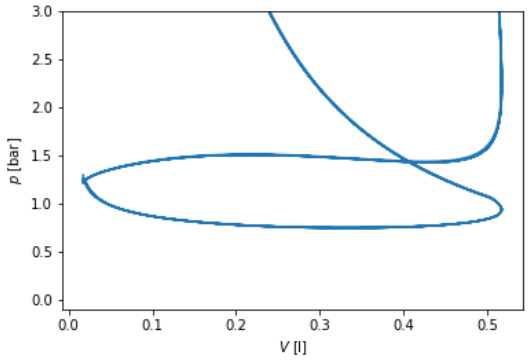

Figure 20: P(V) inlet loop 3500 RPM

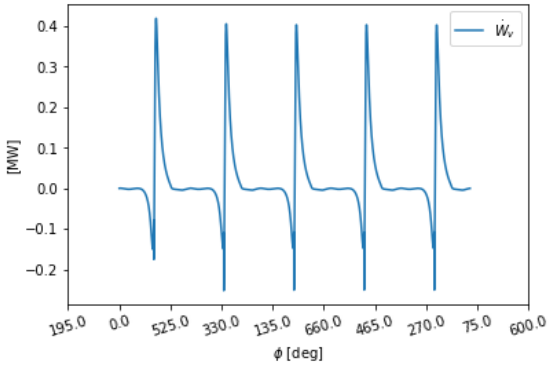

Figure 21: Work 3500 RPM

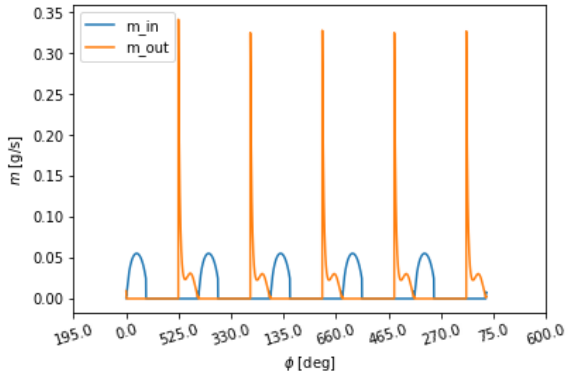

Figure 22: Mass flow 3500 RPM

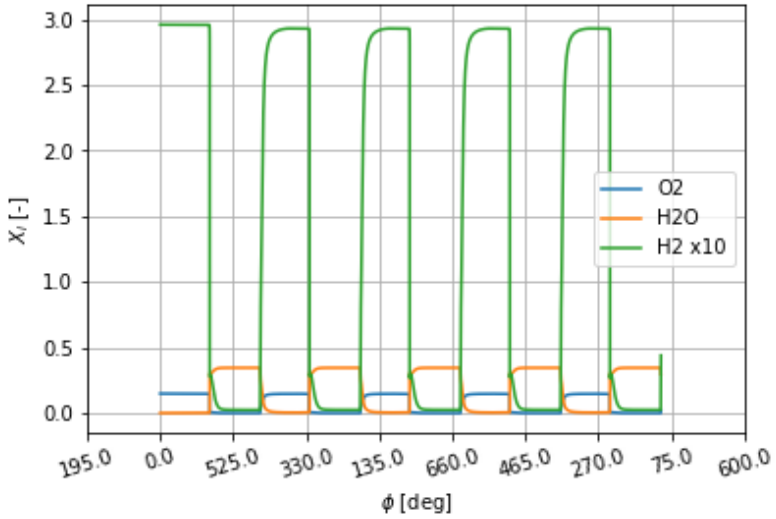

Figure 23: Gas composition 3500 RPM

Plots above refers to case in which valve overlap is applied, which means, that for some time of cycle both inlet and outlet valves are simultaneously open. For case with no valve overlap problems with stable work were observed. This was due to too high initial temperature in cylinder caused by previous cycle which led to instant autoignition of injected fuel mixture as soon as it entered the cylinder. That case could be seen on plots below.

3.4 Case with no valve overlap

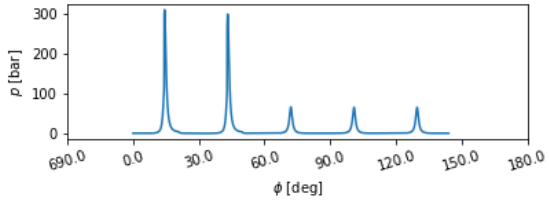

Figure 24: P(phi) 2500 RPM NVO

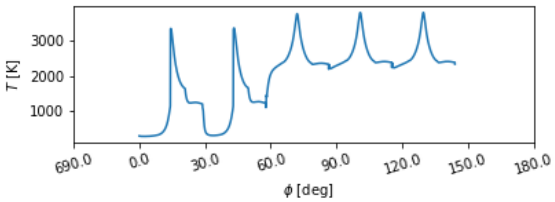

Figure 25: T(phi) 2500 RPM NVO

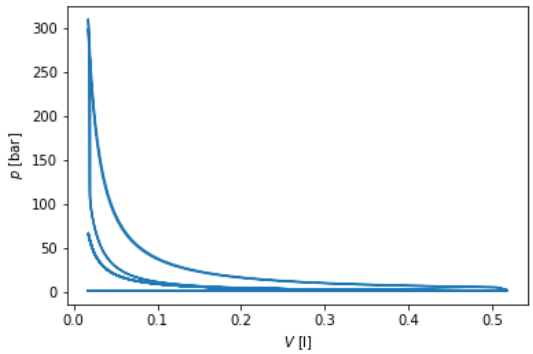

Figure 26: P(V) 2500 RPM NVO

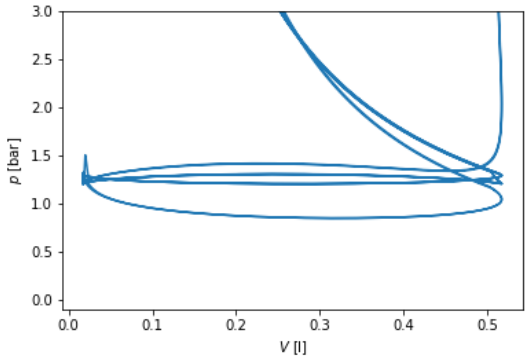

Figure 27: P(V) inlet loop 2500 RPM NVO

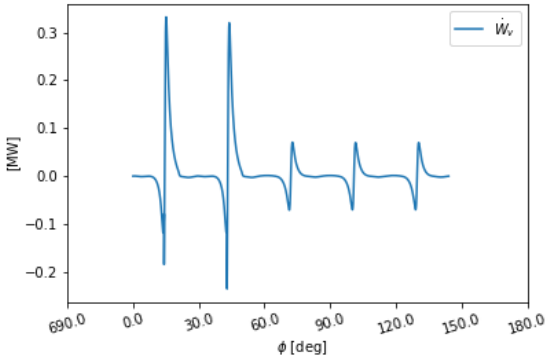

Figure 28: Work 2500 RPM NVO

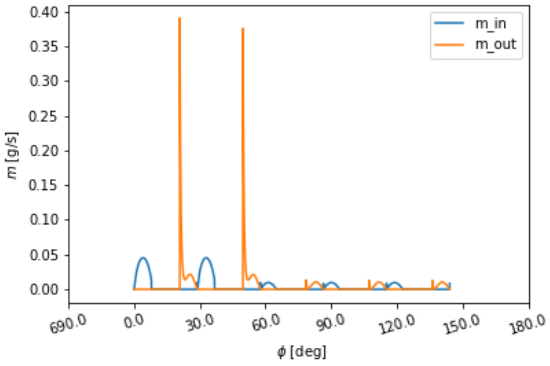

Figure 29: Mass flow 2500 RPM NVO

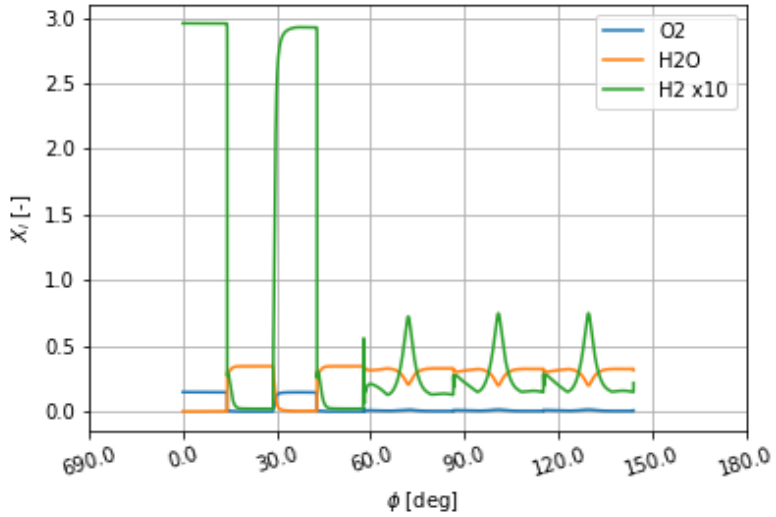

Figure 30: Gas composition 2500 RPM NVO

3.5 Summary

RPM (Valve Overlap [deg])	Efficiency [%]	Expansion Power [kW]	Heat Release Rate [kW]	Comments
2500 (0 deg)	43.9	6.9	15.8	During work, the mixture ignites prematurely
1500 (4 deg)	55.3	11.5	20.9	
2500 (4 deg)	54.9	17.2	31.4	
3500 (4 deg)	54.4	21.6	39.7	

Problem has been observed with temperature distribution in function of cycle (crank angle). Maximum operating temperature reaches over 3000K, which for considering pure hydrogen as a fuel could be acceptable. This high temperature leads to high temperature loads on engine components, which further means their shorter durability and higher mass. Such high temperature would not be achieved when using gasoline as a fuel in HCCI.

4 Conclusions

Although simulations showed that making HCCI engine based on gaseous fuel is theoretically possible, practically it is very hard to implement. The reason is because gaseous fuels need very high compression ratios (>30) to ignite and such compression ratios are unreachable with nowaday technology. In this case SPCCI (Spark Plug Controlled Compression Ignition) engine would be more suitable, but it is not the case of this paper.

References

- *https://cantera.org/examples/python/index.html*
- *Zhao, Fuquan; Asmus, Thomas W.; Assanis, Dennis N.; Dec, John E.; Eng, James A.; Najt, Paul M. (2003). Homogeneous Charge Compression Ignition (HCCI) Engines: Key Research and Development Issues. Warrendale, PA, USA: Society of Automotive Engineers*
- *John E.; Epping, Kathy; Aceves, Salvador M.; Bechtold, Richard L. (2002). "The Potential of HCCI Combustion for High Efficiency and Low Emissions". Society of Automotive Engineers*
- *Stanglmaier, Rudolf (1999). "Homogeneous Charge Compression Ignition (Hcci): Benefits, Compromises, and Future Engine Applications". Society of Automotive Engineers*

List of Figures

Python code

```
"""
Simulation of a Hydrogen fueled Homogenous C Combustion Engine.

The use of pure hydrogen as fuel requires an unrealistically high temperature

"""

import cantera as ct
import numpy as np

#############################################################################
# Input Parameters
#############################################################################

f = 4000. / 60.  # engine speed [1/s] (3000 rpm)
V_H = .5e-3  # displaced volume [m**3]
epsilon = 30.  # compression ratio [-]
d_piston = 0.083  # piston diameter [m]

# turbocharger temperature, pressure, and composition
T_inlet = 300.  # K
p_inlet = 1.3e5  # Pa
comp_inlet = 'H2:2, O2:1, N2:3.76'

# outlet pressure
p_outlet = 1.2e5  # Pa

# ambient properties
T_ambient = 300.  # K
p_ambient = 1e5  # Pa
comp_ambient = 'O2:1, N2:3.76'

# Reaction mechanism name
reaction_mechanism = 'gri30.xml'

# Inlet valve friction coefficient, open and close timings
inlet_valve_coeff = 1.e-6
inlet_open = -2. / 180. * np.pi
inlet_close = 198. / 180. * np.pi

# Outlet valve friction coefficient, open and close timings
outlet_valve_coeff = 1.e-6
outlet_open = 522. / 180 * np.pi
outlet_close = 2. / 180. * np.pi

# Simulation time and resolution
sim_n_revolutions = 10.
sim_n_timesteps = 30000.
```

```
#####################################################################

# load reaction mechanism
gas = ct.Solution(reaction_mechanism)

# define initial state
gas.TPX = T_inlet, p_inlet, comp_inlet
r = ct.IdealGasReactor(gas)

# define inlet state
gas.TPX = T_inlet, p_inlet, comp_inlet
inlet = ct.Reservoir(gas)

# define outlet pressure (temperature and composition don't matter)
gas.TPX = T_ambient, p_outlet, comp_ambient
outlet = ct.Reservoir(gas)

# define ambient pressure (temperature and composition don't matter)
gas.TPX = T_ambient, p_ambient, comp_ambient
ambient_air = ct.Reservoir(gas)

# set up connecting devices
inlet_valve = ct.Valve(inlet, r)
outlet_valve = ct.Valve(r, outlet)
piston = ct.Wall(ambient_air, r)

# convert time to crank angle
def crank_angle(t):
    return np.remainder(2 * np.pi * f * t, 4 * np.pi)

# set up IC engine parameters
V_oT = V_H / (epsilon - 1.)
A_piston = .25 * np.pi * d_piston ** 2
stroke = V_H / A_piston
r.volume = V_oT
piston.area = A_piston
def piston_speed(t):
    return -stroke / 2 * 2 * np.pi * f * np.sin(crank_angle(t))
piston.set_velocity(piston_speed)

# create a reactor network containing the cylinder
sim = ct.ReactorNet([r])

# set up output data arrays
states = ct.SolutionArray(r.thermo)
t_sim = sim_n_revolutions / f
t = (np.arange(sim_n_timesteps) + 1) / sim_n_timesteps * t_sim
V = np.zeros_like(t)
m = np.zeros_like(t)
test = np.zeros_like(t)
```

```
mdot_in = np.zeros_like(t)
mdot_out = np.zeros_like(t)
d_W_v_d_t = np.zeros_like(t)

# set parameters for the automatic time step refinement
n_last_refinement = -np.inf  # for initialization only
n_wait_coarsening = 10

# do simulation
for n1, t_i in enumerate(t):
    # define opening and closing of valves and injector
    if (np.mod(crank_angle(t_i) - inlet_open, 4 * np.pi) <
            np.mod(inlet_close - inlet_open, 4 * np.pi)):
        inlet_valve.set_valve_coeff(inlet_valve_coeff)
        test[n1] = 1
    else:
        inlet_valve.set_valve_coeff(0)

    if (np.mod(crank_angle(t_i) - outlet_open, 4 * np.pi) <
            np.mod(outlet_close - outlet_open, 4 * np.pi)):
        outlet_valve.set_valve_coeff(outlet_valve_coeff)
    else:
        outlet_valve.set_valve_coeff(0)

    # perform time integration, refine time step if necessary
    solved = False
    for n2 in range(4):
        try:
            sim.advance(t_i)
            solved = True
            break
        except ct.CanteraError:
            sim.set_max_time_step(1e-6 * 10. ** -n2)
            n_last_refinement = n1
    if not solved:
        raise ct.CanteraError('Refinement limit reached')
    # coarsen time step if too long ago
    if n1 - n_last_refinement == n_wait_coarsening:
        sim.set_max_time_step(1e-5)

    # write output data
    states.append(r.thermo.state)
    V[n1] = r.volume
    m[n1] = r.mass
    mdot_in[n1] = inlet_valve.mdot(0)
    mdot_out[n1] = outlet_valve.mdot(0)
    d_W_v_d_t[n1] = - (r.thermo.P - ambient_air.thermo.P) * A_piston * \
        piston_speed(t_i)


print('P_max:\t'+format(max(states.P/1.e5),'2.1f')+'bar')
```

```
print('T_max:\t'+format(max(states.T),'2.1f')+'K')
print('W_max:\t'+format(max(d_W_v_d_t/1.e6),'2.1f')+'MJ')

#########################################################################
# Plot Results in matplotlib
#########################################################################

import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
import numpy

phi=np.zeros_like(t)
phi=crank_angle(t) * 180 / np.pi

# pressure and temperature
plt.clf()
plt.subplot(211)
plt.plot(t, states.P / 1.e5)
plt.ylabel('$p$ [bar]')
plt.xlabel('$\phi%.3$ [deg]')
plt.xticks(plt.xticks()[0], numpy.round(crank_angle(plt.xticks()[0]) * 180 / n
plt.savefig('ic_engine_P_(phi).png')
plt.show()

plt.subplot(212)
plt.plot(t, states.T)
plt.ylabel('$T$ [K]')
plt.xlabel('$\phi%.3$ [deg]')
plt.xticks(plt.xticks()[0], numpy.round(crank_angle(plt.xticks()[0]) * 180 / n
plt.savefig('ic_engine_T_(phi).png')
plt.show()

# inlet and outlet mass flow
plt.clf()
plt.plot(t, mdot_in, label='m_in')
plt.plot(t, mdot_out, label='m_out')
plt.ylabel('$m$ [g/s]')
plt.xlabel('$\phi$ [deg]')
plt.legend(loc=0)
plt.xticks(plt.xticks()[0], numpy.round(crank_angle(plt.xticks()[0]) * 180 / n
           rotation=17)
plt.savefig('ic_engine_In_Out_mass.png')
plt.show()

# p-V diagram
plt.clf()
plt.plot(V[t<2*4*np.pi/f] * 1000, states.P[t<2*4*np.pi/f] / 1.e5)
plt.xlabel('$V$ [l]')
plt.ylabel('$p$ [bar]')
plt.savefig('ic_engine_P_(V).png')
plt.show()
```

```
# p-V diagram; inlet/outlet loop
plt.clf()
plt.plot(V[t<2*4*np.pi/f] * 1000, states.P[t<2*4*np.pi/f] / 1.e5)
plt.xlabel('$V$ [l]')
plt.ylim(-0.1, 3.)
plt.ylabel('$p$ [bar]')
plt.savefig('ic_engine_P_(V)_inlet_outlet_loop.png')
plt.show()

# expansion work
plt.clf()
plt.plot(t, d_W_v_d_t/1e6, label='$\dot{W}_v$')
plt.legend(loc=0)
plt.ylabel('[MW]')
plt.xlabel('$\phi$ [deg]')
plt.xticks(plt.xticks()[0], numpy.round(crank_angle(plt.xticks()[0]) * 180 / n
           rotation=17)
plt.savefig('ic_engine_Q_W.png')
plt.show()

# gas composition
plt.clf()
plt.plot(t, states('O2').X, label='O2')
plt.plot(t, states('H2O').X, label='H2O')
plt.plot(t, states('H2').X * 10, label='H2 x10')
plt.legend(loc=0)
plt.grid(True)
plt.ylabel('$X_i$ [-]')
plt.xlabel('$\phi$ [deg]')
plt.xticks(plt.xticks()[0], numpy.round(crank_angle(plt.xticks()[0]) * 180 / n
           rotation=17)
plt.savefig('ic_engine_gas_comp.png')
plt.show()



#########################################################################
# Integral Results
#########################################################################

from scipy.integrate import trapz

# Total mass flow through inlet and oulet valve
fr_m_in=trapz(mdot_in, t)
fr_m_out=trapz(mdot_out, t)

# Difference between Inlet Flow and Outlet Flow; should be ~0
fr_diff=fr_m_in-fr_m_out

#Reactants Enthalpy
```

```
cv=r.thermo.cv_mass
print(cv)
#Products Enthalpy


#Difference between Inlet Enthalpy and Outlet Enthalpy


#Higher heating value for H2 [J/kg]
H2_hhv=120.1*1e6
#Total Heat Release
ni_H2=2
ni_O2=32
ni_N2=28
heat_release=(fr_m_in)*((2*ni_H2)/(2*ni_H2+1*ni_O2+3.76*ni_N2))*H2_hhv

#Total Heat Release and Expantion Work
Q = heat_release
W = trapz(d_W_v_d_t, t)

# Efficency
eta = W/Q

# Results
print('Heat release rate per cylinder (estimate):\t' + format(Q / t_sim / 1000
print('Expansion power per cylinder (estimate):\t' + format(W / t_sim / 1000.,
print('Efficiency (estimate):\t\t\t' + format(eta * 100., ' 2.1f') + ' %')
print('Inlet Mass Flow (estimate):\t\t\t' + format(fr_m_in * 1000., ' 2.3f') +
print('Outlet Mass Flow (estimate):\t\t\t' + format(fr_m_out * 1000., ' 2.3f')
print('Outlet Mass Flow (estimate):\t\t\t' + format(fr_diff * 1000., ' 2.3f')
```