Trabalho 1 - Sistema de travagem ABS

14 de dezembro 2022

André Oliveira Barbosa 91684

Francisco António Borges Paulino a91666

Caso de estudo

No contexto do sistema de travagem ABS (“Anti-Lock Breaking System”), pretende-se construir um autómato híbrido que descreva o sistema e que possa ser usado para verificar as suas propriedades dinâmicas.

1. A componente discreta do autómato contém os modos: `Start` , `Free` , `Stopping` , `Blocked` , e `Stopped` . No modo `Free` não existe qualquer força de travagem; no modo `Stopping` aplica-se a força de travagem alta; no modo `Blocked` as rodas estão bloqueadas em relação ao corpo mas o veículo move-se (i.e. derrapa); no modo `Stopped` o veículo está imobilizado.
2. A componente contínua do autómato usa variáveis contínuas V, v para descrever a `velocidade do corpo` e a velocidade linear das rodas ambas em relação ao solo.
3. Assume-se que o sistema de travagem exerce uma força de atrito proporcional à diferença das duas velocidades. A dinâmica contínua, as equações de fluxo, está descrita abaixo.
4. Os “switchs” são a componente de projeto deste trabalho; cabe ao aluno definir quais devem ser de modo a que o sistema tenha um comportamento desejável: imobilize-se depressa e não “derrape” muito.
5. É imprescindível evitar que o sistema tenha “trajetórias de Zenão”. Isto é, sequências infinitas de transições entre dois modos em intervalos de tempo que tendem para zero mas nunca alcançam zero.

Objetivos

1. Definir um autómato híbrido que descreva a dinâmica do sistema segundo as notas abaixo indicadas e com os “switchs” por si escolhidos.
2. Modelar em lógica temporal linear LT propriedades que caracterizam o comportamento desejável do sistema. Nomeadamente

    `i. o veículo imobiliza-se completamente em menos de t segundos`
    `ii. a velocidade V diminui sempre com o tempo`

3. Codificar em SMT’s o modelo definido em 1.
4. Codificar em SMT’s a verificação das propriedades temporais definido em 2.

In [58]:

```
import matplotlib.pyplot as plt
```

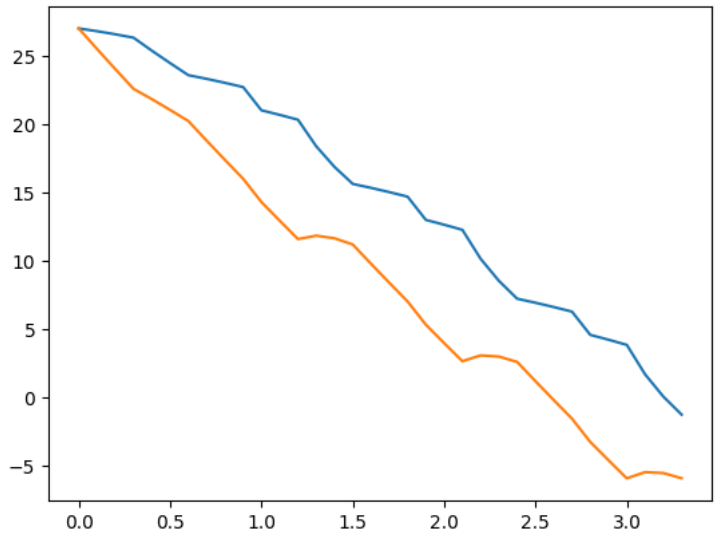
Plot

In [59]:

```
def constantes_plot(a, b, c, P, time, v_inicial, epsilon):
    v = v_inicial
    r = v_inicial
    t = 0
    V = [v]
    R = [r]
    T = [t]
    dt = 0.1
    x = 0.3
    timer = 0
    m = "free"
    while(t<time and (v>0 or r>0)):
        if v <= 0.1 :
            m = "stopped"
        print(str(t)+':'+m)
        if timer > x and m== "free":
            c = 2
            m = "stopping"
            timer = 0
        elif m=="stopping" and ( (v - r) <= epsilon or timer > x) :
            c = 0.2

            m = "blocked"
            timer = 0
        elif timer >  x and m == "blocked":
            #c = 0.1
            m = "free"
            timer = 0
            var = (-a*P-b)*dt
            v  += var
            r += var
            timer += dt
            t += dt
            V.append(v)
            R.append(r)
            T.append(t)
            continue
        timer += dt
        v,r  = v +(-c*(v-r)-b)*dt, r + (-a*P + c *(v-r))*dt
        t += dt
        V.append(v)
        R.append(r)
        T.append(t)
    m = "stopped"
    plt.plot(T,V,T,R)
constantes_plot(0.01, 2, 0.2, 1500, 20, 27, 0.5)
```

```
0:free
0.1:free
0.2:free
0.30000000000000004:free
0.4:stopping
0.5:stopping
0.6:stopping
0.7:blocked
0.7999999999999999:blocked
0.8999999999999999:blocked
0.9999999999999999:free
1.0999999999999999:free
1.2:free
1.3:stopping
1.4000000000000001:stopping
1.5000000000000002:stopping
1.6000000000000003:blocked
1.7000000000000004:blocked
1.8000000000000005:blocked
1.9000000000000006:free
2.0000000000000004:free
2.1000000000000005:free
2.2000000000000006:stopping
2.3000000000000007:stopping
2.400000000000001:stopping
2.500000000000001:blocked
2.600000000000001:blocked
2.700000000000001:blocked
2.800000000000001:free
2.9000000000000012:free
3.0000000000000013:free
3.1000000000000014:stopping
3.2000000000000015:stopped
```



In [60]:

```
import matplotlib.pyplot as plt
from z3 import *
Mode, (START,STOPPING,STOPPED,BLOCKED,FREE) = EnumSort('Mode', ('START','STOPPING','STOPPED','BLOCKED','FREE'))

def declare(i):
    state = {}
    state['Vr'] = Real('Vr'+str(i))
    state['Vc'] = Real('Vc'+str(i))
    state['t']  = Real('t' +str(i))
    state['m']  = Const('m'+str(i),Mode)
    state['timer'] = Real('timer'+str(i))
    return state
```

Transições

Para assegurar o comportamento desejado do sistema ABS definimos as seguintes transições, visíveis no seguinte autómato:

UNTIMED

```
Start → Free ,

Stopping → Stopped

Free → Stopped

Blocked → Stopped

Free → Stopping

Stopping → Blocked

Blocked → Free
```

Timed

```
Free → Free ,

Stopping → Stopping

Blocked → Blocked

Stopped → Stopped
```

In [61]:

```
a, b, c1,c2, P, tau,intervalo = 0.01, 0.5, 0.5,10, 1000, 0.1,0.1

def init(s, v_input, a, b, c):
  #t -> tempo, m-> modo, v-> velocidade rodas, V -> velocidade corpo
  return And(s['t'] == 0, s['m'] == START, s['Vr'] == v_input, s['Vc'] == v_input)

def trans(s, p):

    stopping_stopped = And( s['t']  == p['t'],
                            s['m']  == STOPPING,
                            p['m']  == STOPPED,
                            s['Vc'] <= 0.1,
                            s['Vr'] <= 0.1,
                            p['Vc'] == 0,
                            p['Vr'] == 0,
                            p['timer'] == 0
                          )

    free_stopped    =  And( s['t'] == p['t'],
                            s['m'] == FREE,
                            p['m'] == STOPPED,
                            s['Vc'] <= 0.1,
                            s['Vr'] <= 0.1,
                            p['Vc'] == 0,
                            p['Vr'] == 0,
                            p['timer'] == 0
                          )

    start_free       = And( s['t'] == p['t'],
                            s['Vr'] == p['Vr'],
                            s['Vc'] == p['Vc'],
                            s['m'] == START,
                            p['m'] == FREE,
                            s['timer'] == 0,
                            p['timer'] == 0 )

    free_stopping    = And( s['t'] == p['t'],
                            s['Vr'] == p['Vr'],
                            s['Vc'] == p['Vc'],
                            s['m'] == FREE,
                            p['m'] == STOPPING,
                            p['timer']==0,
                            s['Vr'] > 0.1,
                            Or(s['timer']>=tau,s['Vc'] - s['Vr']>=0.1)
                          )

    stopping_blocked = And( s['t'] == p['t'],
                            s['Vr'] == p['Vr'],
                            s['Vc'] == p['Vc'],
                            s['m'] == STOPPING,
                            p['m'] == BLOCKED,
                            s['Vc'] < s['Vr'] + 0.2,
                            s['timer'] == 0,
                            p['timer'] == 0,
                            s['Vr'] > 0.1
                          )

    blocked_stopped = And(  s['t'] == p['t'],
                            s['m'] == BLOCKED,
                            p['m'] == STOPPED,
                            s['Vc'] <= 0.1,
                            s['Vr'] <= 0.1,
                            p['Vc'] == 0,
                            p['Vr'] == 0,
                            p['timer'] == 0
                         )

    blocked_free     = And( s['t'] == p['t'],
                            s['Vr'] == p['Vr'],
                            s['Vc'] == p['Vc'],
                            s['m'] == BLOCKED,
                            p['m'] == FREE,
                            s['timer']>=tau,
                            p['timer']==0,
                            s['Vr'] > 0.1
                          )




  #timed

    free_free = And(s['m'] == FREE,
                    p['m'] == FREE,
                    p['Vc']   == s['Vc'] +   (-c1 * (s['Vc'] - s['Vr']) -b) * intervalo,
                    p['Vr']   == s['Vr'] +   (-a * P + c1 * (s['Vc'] - s['Vr'])) * intervalo,
                    p['t'] == s['t']+intervalo,
```

```
                    p['timer']==s['timer']+intervalo,

                    p['timer']<=tau,
                    s['Vr'] > 0.1,
                   )

    stopping_stopping = And(s['m'] == STOPPING, p['m'] == STOPPING,
                            p['Vc']   == s['Vc'] +   (-c2 * (s['Vc'] - s['Vr']) - b) * intervalo,
                            p['Vr']   == s['Vr'] +   (-a * P + c2 * (s['Vc'] - s['Vr'])) * intervalo,
                            s['Vc'] > s['Vr'] + 0.2,
                            p['t'] == s['t']+intervalo,
                            s['timer'] == p['timer'],
                           )

    blocked_blocked = And(s['m'] == BLOCKED, p['m'] == BLOCKED,

                          p['Vc']   == s['Vc'] + (-a*P-b)*intervalo,
                          p['Vr']   == s['Vr'] + (-a*P-b)*intervalo,
                          p['t'] == s['t']+intervalo,
                          p['timer']==s['timer']+intervalo,
                          p['timer']<=tau,
                         )

    stopped_stopped = And(s['m'] == STOPPED,
                          p['m'] == STOPPED,
                          p['t'] == s['t']+intervalo,
                          s['timer'] == 0,
                          p['timer'] == 0,
                          s['Vr'] == p['Vr'],
                          s['Vc'] == p['Vc'])

    return Or( stopping_blocked,stopping_stopped,free_stopped, blocked_stopped,start_free, free_stopping, blocked_free, free_free, stoppir
```

In [62]:

```
def frac2float(x):
    #print(x)
    return float(x.numerator_as_long())/float(x.denominator_as_long())
def gera_traco(declare, init, trans, k):
    T=[]
    VV=[]
    VR=[]
    s = Solver()
    traco = [declare(i) for i in range(k)]

    s.add(init(traco[0], 27,0.01,0.1,0.5))

    for i in range(k-1):
        s.add(trans(traco[i], traco[i+1]))

    if s.check() == sat:
        m = s.model()

        T  = [frac2float(m[traco[i]["t"]] ) for i in range(k)]
        VV = [frac2float(m[traco[i]["Vc"]]) for i in range(k)]
        VR = [frac2float(m[traco[i]["Vr"]]) for i in range(k)]
        for i in range(k):
            print("Estado:", i)
            for v in traco[i]:

                res = m[traco[i][v]]
                #print(res)
                #print



                if res.sort() != RealSort():
                    print(v, '=', res)
                else:
                    print(v, '=', float(res.numerator_as_long())/float(res.denominator_as_long()))
            print()
    else:
        print("Não tem solução.")
    plt.plot(T, VV, T, VR)
```

Gera_Traco

In [63]:

```
gera_traco(declare, init, trans,80)
```

```
Estado: 0
Vr = 27.0
Vc = 27.0
t = 0.0
m = START
timer = 0.0

Estado: 1
Vr = 27.0
Vc = 27.0
t = 0.0
m = FREE
timer = 0.0

Estado: 2
Vr = 26.0
Vc = 26.95
t = 0.1
m = FREE
```

Propriedades

De seguida encontra-se a definição das propriedes i e ii definidas acima no objetivo 2

In [64]:

```
def imobXSec(state):
    return Implies(state['t']>=4,state['m']==STOPPED)
def sempreMenor(atual,prox):
    return Implies(And(atual['m']==STOPPED,atual['t']<prox['t']),atual['Vc']>prox['Vc'])
```

In [65]:

```
def testaImob(declare,init,trans,inv,K):
    for k in range(1,K+1):
        s = Solver()

        trace = [declare(i) for i in range(k)]

        s.add(init(trace[0],27,0.01,0.1,0.5))
        for i in range(k-1):
            s.add(trans(trace[i],trace[i+1]))

        s.add(Not(inv(trace[k-1])))

        if s.check() == sat:
            m = s.model()
            for i in range(k):
                for v in trace[i]:
                    print(i,v,'=',m[trace[i][v]])
            return

    print ("Property MAY be valid")
```

In [66]:

```
testaImob(declare,init,trans,imobXSec,50)
```

```
Property MAY be valid
```

In [67]:

```
def testaMenor(declare,init,trans,inv,K):
    for k in range(1,K+1):
        s = Solver()

        trace = [declare(i) for i in range(k)]

        s.add(init(trace[0],27,0.01,0.1,0.5))
        aux=[]
        for i in range(k-1):
            s.add(trans(trace[i],trace[i+1]))
            aux.append(Not(inv(trace[i],trace[i+1])))

        s.add(Or(aux))

        if s.check() == sat:
            m = s.model()
            for i in range(k):
                for v in trace[i]:
                    print(i,v,'=',m[trace[i][v]])
            return

    print ("Property MAY be valid")
```

In [68]:

```
testaMenor(declare,init,trans,sempreMenor,20)
```

```
Property MAY be valid
```

In [ ]: