Motion Planning HW4 report

Xie Fujing

2023.02.03

1. Code completion:

```
//p=p_0+v_0*t+0.5*a*t^2
//v=v_0+a*t          You, 前天 • Uncommitted changes
pos+=vel*delta_time+0.5*delta_time*delta_time*acc_input;
vel+=delta_time*acc_input;
```

Figure 1: task1

```
double diff_px=_target_position(0)-_start_position(0);
double diff_py=_target_position(1)-_start_position(1);
double diff_pz=_target_position(2)-_start_position(2);
double delta_vx=0;
double delta_vy=0;
double delta_vz=0;
double v0x=_start_velocity(0);
double v0y=_start_velocity(1);
double v0z=_start_velocity(2);
//coefficient of quartic equation, which is the Derivative of J
double coff_a=1;
double coff_b=0;
double coff_c=4.0*delta_vx*delta_vx- 12.0*delta_vx*v0x - 4.0*delta_vy*delta_vy - 12.0*delta_vy*v0y - 4.0*delta_vz*delta_vz -
12.0*delta_vz*v0z - 12.0*v0x*v0x - 12.0*v0y*v0y - 12.0*v0z*v0z;
double coff_d= 24.0*delta_vx*diff_px + 24.0*delta_vy*diff_py + 24.0*delta_vz*diff_pz + 48.0*diff_px*v0x + 48.0*diff_py*v0y + 48.0*diff_pz*v0z ;
double coff_e= - 36.0*diff_px*diff_px - 36.0*diff_py*diff_py - 36.0*diff_pz*diff_pz;

std::complex<double> answer[4];
//use ferrari to get the zero root of Derivative of J
Ferrari(answer,coff_a,coff_b,coff_c,coff_d,coff_e);
std::complex<double> z1 {0,1e-10};
double argminT;
//filter real and positive root of quartic equation
for(int i=0;i<4;i++){
  if(abs(answer[i].imag())<z1.imag()){
    if(answer[i].real()>z1.real()){
      argminT=answer[i].real();
      // cout<<"answer "<<i<<"="<<answer[i]<<endl;
    }
  }
}
double alpha1 = 12 * (-diff_px + argminT * v0x) / argminT / argminT / argminT + 6 *delta_vx / argminT / argminT;
double alpha2 = 12 * (-diff_py + argminT * v0y) / argminT / argminT / argminT + 6 * delta_vy / argminT / argminT;
double alpha3= 12 * (-diff_pz + argminT * v0z) / argminT / argminT / argminT+ 6 *delta_vz / argminT / argminT;
double beta1 = 6 *(-diff_px + argminT * v0x) / argminT / argminT-2 *delta_vx / argminT ;
double beta2 =  6 *(-diff_py + argminT * v0y) / argminT / argminT-2 *delta_vy / argminT ;
double beta3 =  6 *(-diff_pz + argminT * v0z) / argminT / argminT-2 *delta_vz / argminT ;

double minJ=argminT+(1.0/3.0*alpha1*alpha1*argminT*argminT*argminT+alpha1*beta1*argminT*argminT+beta1*beta1*argminT)+
              (1.0/3.0*alpha2*alpha2*argminT*argminT*argminT+alpha2*beta2*argminT*argminT+beta2*beta2*argminT)+
              (1.0/3.0*alpha3*alpha3*argminT*argminT*argminT+alpha3*beta3*argminT*argminT+beta3*beta3*argminT);
return minJ;
```

Figure 2: task2

2. Get J's derivative in terms of T:
   With the help from python package 'sympy', the J's derivative expression in terms of T can be obtained. Code is attached below, script is located in 'src' folder of package 'grid__path__searcher' named 'solveEquation.py'.

```python
from sympy import *
# This script is used to get the derivative of J
T = symbols('T')
# diff_p?=p_?f-p_?0
diff_px= symbols('diff_px')
diff_py = symbols('diff_py')
diff_pz = symbols('diff_pz')
# v0?=v_?f-v_?0
v0x=symbols('v0x')
v0y=symbols('v0y')
v0z=symbols('v0z')

delta_px=diff_px-v0x*T
delta_py=diff_py-v0y*T
delta_pz=diff_pz-v0z*T

delta_vx = symbols('delta_vx')
delta_vy = symbols('delta_vy')
delta_vz = symbols('delta_vz')
# alfa expression
a1=-12/T**3*delta_px+6/T**2*delta_vx
a2=-12/T**3*delta_py+6/T**2*delta_vy
a3=-12/T**3*delta_pz+6/T**2*delta_vz
# beta expression
b1=6/T**2*delta_px-2/T*delta_vx
b2=6/T**2*delta_py-2/T*delta_vy
b3=6/T**2*delta_pz-2/T*delta_vz
# J expression
J=T+(1/3*a1**2*T**3+a1*b1*T**2+b1**2*T)+(1/3*a2**2*T**3+a2*b2*T**2+b2**2*T)+(1/3*a3**2*T**3+a3*b3*T**2+b3**2*T)
# get J's derivative
J_diff=J.diff(T)
print("J_diff = ")
print( J_diff)
print("--------------------------------------")
J_diff_SIM= simplify(J_diff)
print(J_diff_SIM)
```

Figure 3: Get J's derivative

The result of running this script is shown as below, which gives the cofficient of quartic equation that needs to be solved in next step:

```
------------------------------------------------
1.0*(1.0*T**4 - 4.0*T**2*delta_vx**2 - 12.0*T**2*delta_vx*v0x - 4.0*T**2*delta
_vy**2 - 12.0*T**2*delta_vy*v0y - 4.0*T**2*delta_vz**2 - 12.0*T**2*delta_vz*v0
z - 12.0*T**2*v0x**2 - 12.0*T**2*v0y**2 - 12.0*T**2*v0z**2 + 24.0*T*delta_vx*d
iff_px + 24.0*T*delta_vy*diff_py + 24.0*T*delta_vz*diff_pz + 48.0*T*diff_px*v0
x + 48.0*T*diff_py*v0y + 48.0*T*diff_pz*v0z - 36.0*diff_px**2 - 36.0*diff_py**
2 - 36.0*diff_pz**2)/T**4
------------------------------------------------
```

Figure 4: J's derivative

3. Solve the quartic equation to get T that minimize J:
   The quartic equation is solved by Ferrari's solution, which is explained here:
   https://en.wikipedia.org/wiki/Quartic_equation#Ferrari's_solution.
   The code to solve the quartic equation using Ferrari's_solution is from
   https://baike.baidu.com/item/一元四次方程/5945955,
   which is in function "std::complex<double> Homeworktool::Ferrari(std::complex<double> x[4],std::complex<double>a,std::complex<double>b,std::complex<double> c,std::complex<double> d,std::complex<double> e)"
4. After get the argmin T, the minimum J can be obtained by equation in the document.
5. Running result:

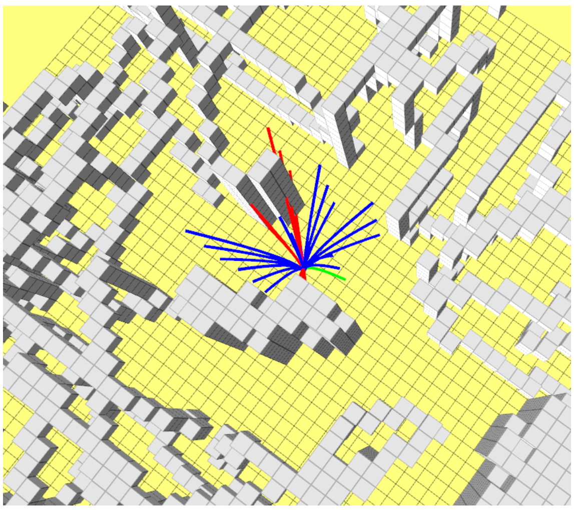

Figure 5: Running result