TITAN installation

1) Create a conda/miniconda environment with python 3.10

wget https://repo.anaconda.com/archive/Anaconda3-2023.03-1-Linux-x86_64.sh
bash ….

Add channels to conda to download the libraries

conda config --add channels conda-forge
conda config --add channels open3d-admin

2) Clone TITAN repository and checkout to the relevant branch:

git clone https://github.com/strath-ace/TITAN.git
git checkout ESA-detection-PATO-dev-ablation

3) Install mamba)
conda install -c conda-forge mamba=1.4.4

4) Install libraries using mamba
mamba install fenics=2019.1.0=py310hff52083_34 (build working)

5) Install remaining libraries using pip
pip3 install numpy==1.25.0; pip3 install pandas==1.5.3; pip3 install meshio==5.3.4; pip3 install scipy==1.10.1; pip3 install pyglet==1.5.27; pip3 install vtk==9.2.6; pip3 install sympy==1.12; pip3 install pymap3d==3.0.1; pip3 install lxml==4.9.4; pip3 install bs4; pip3 install scikit-build==0.17.6; pip3 install pytest; pip install open3d==0.17.0; pip install gmsh==4.10.5; pip install python-fcl; pip install trimesh; pip3 install h5py; pip3 install embreex; pip3 install vg

6) Install library for unit testing
mamba install pytest

- To run pytest, use python -m pytest -s Tests/Regression at TITAN root folder

7) Install mutation++ library for anaconda:
https://github.com/mutationpp/Mutationpp/blob/master/docs/installation.md#top

- git clone repository
- run cmake
- (clone pybind at th thirdparty folder)
- install python api for mutationpp
    - mamba install scikit-build=0.17.6
    - root folder → python setup.py build (root folder i.e. /Mutationpp)
    - python setup.py install

8) Install SU2

- git clone
- (git checkout f4fee7fece8eaeadc4e7be1f1d2be570a251028e)
- meson.py and ninja
- add LD_libary_path of mutation to the bashrc file

9) for PATO coupling:
https://github.com/nasa/pato

- git clone repository
- for installation, follow instruction in PATO documentation
- copy modified files from TITAN folder into PATO folder:

    $ cp TITAN/Thermal/HeatFluxBoundaryConditions.*
    PATO/src/applications/libraries/libPATOx/MaterialModel/BoundaryConditions/HeatFlux/

    $cp TITAN/Thermal/threeDtecplotBoundaryMappingModel.*
    src/applications/libraries/libPATOx/MaterialModel/BoundaryConditions/BoundaryMapping/
    types/3D/

    $ cp TITAN/Thermal/FourierMaterialPropertiesModel.*
    src/applications/libraries/libPATOx/MaterialModel/MaterialPropertiesModel/types/Fourier/

    $ cp BprimeBoundaryConditions.*
    PATO/src/applications/libraries/libPATOx/MaterialModel/BoundaryConditions/Bprime/

- recompile PATO

TITAN I/O

CODE INPUTS

1. TITAN input file

The user can find a working example for an ATV-1 trajectory simulation in Tests/Configs/ATV.txt.

In this section, each input of the code is explained. All inputs must be defined in SI units unless stated otherwise (only for angle inputs).

[Options]
Num_iters – number of iterations to run the simulations
Load_mesh – if a previous simulation has already been performed with the current geometry and the output folder name has not changed/has not been deleted, setting this flag to True allows to run a new simulation from start, however skipping the stage at which the geometry is pre-processed
Load_state - if a previous simulation has already been performed with the current geometry and input conditions and the output folder name has not changed/has not been deleted, setting this flag to True allows to restart the simulation from the last iteration
Fidelity – defines the fidelity of the simulation ('low', 'high', 'multi'). For this project, only 'low' should be used
Output_folder – output folder name
Output_freq – Visualization output is created every n iterations
Save_freq – Restart solutions are created every n iterations

[Time]
Time_step – time-step of the simulation in seconds
Time_integration – time discretization scheme (only 'euler' is available)

[Trajectory]
Altitude – initial altitude in meters
Velocity – initial velocity in m/s
Flight_path_angle – initial flight path angle in degrees
Heading_angle – initial heading angle in degrees
Latitude – initial latitude in degrees
Longitude – initial longitude in degrees

[Thermal]
Ablation – flag to switch on ablation modelling
Ablation_mode – defines thermal model to be used for surface temperature calculation

[PATO]
N_cores – number of cores for the PATO simulation (needs to be at least 2 cores)
Time_step – time-step of the PATO simulation
Conduction_objects – flag defining whether heat conduction modelling between different objects is modelled

[Radiation]

Frequency – Frequency of iterations to post-process radiative emissions (since it's a post-processing script, can only calculate emissions for available solutions, set in Save_freq)

[Model]

Planet – Planet atmosphere to be simulated (only 'Earth' available for this project)

[Aerothermo]

Heat_model – model used for low-fidelity convetive heat flux calculation (sc – Scarab formulation and Lees distribution; vd – Van Driest; fr – Fay-Riddell; sq – Sutton Graves)

[Assembly]

Path – system path to the folder where the geometry is contained

Connectivity – connectivity between the different .stl components of the geometry. To connect components X and Y use [X, Y, 0]

Angle_of_attack – initial angle of attack of the assembly in degrees

Sideslip – initial sideslip angle of the assembly in degrees

Roll – initial roll angle of the assembly in degrees

[Objects]

The list of objects/components takes 2 types: **Primitive** and Joint.

Each objects takes as input: .stl surface mesh file, component type, initial temperature, material, fenics_id (always 1 for this project), volume mesh generation parameters for BLOOM tool, fragmentation trigger type (altitude, iteration, temperature, time) and fragmentation trigger value.

If **Primitive**:

<Component_name> = [<component>.stl, TYPE = Primitive, TEMPERATURE = <initial_temperature>, MATERIAL = <material_from_database>, FENICS_ID = 1, BLOOM = (<boolean_flag>; <number_of_prismatic_layers>; <1st_layer_height>; <growth_factor>), TRIGGER_TYPE = <trigger_type>, TRIGGER_VALUE = <trigger_value>]

<Component_name> = [<component>.stl, TYPE = Joint, TEMPERATURE = <initial_temperature>, MATERIAL = <material_from_database>, FENICS_ID = 1, BLOOM = (<boolean_flag>; <number_of_prismatic_layers>; <1st_layer_height>; <growth_factor>), TRIGGER_TYPE = <trigger_type>, TRIGGER_VALUE = <trigger_value>]

[Initial Conditions]

Rotation = (1: [0,0,0]) - initial angle of the vehicle in the body frame (the same reference frame in which the .stl geometry file is provided) in degrees

Angular Velocity = (1 : [0,0,0]) - initial angular velocity of the vehicle in the body frame (the same reference frame in which the .stl geometry file is provided) in degrees

2. Radiative emissions

The post-processing script to calculate radiative blackbody and atomic emissions is located in TITAN/Postprocess/postprocess_emissions.py. The script is tailored to calculate Aluminum I and Oxygen I emissions, for the wavelenghts associated to the respective atomic spectral lines.

If the user wishes to calculate emissions for different frequencies, the user should edit the function ***emissions(…)*** in file ***postprocess_emissions.py*** and replace the hardcoded wavelenghts with the desired values.

The explicit calculation of emissions for Al I and O I is implemented in the file TITAN/Thermal/thermal.py. If

the user wishes to calculate emissions for different chemical elements, besides adjusting the wavelengths accordingly, the user must also create a new function for the purpose, following the implementation provided for Al I and O I, and editing the chemical element data accordingly.

3. Other input-related guidelines

- The user must provide one .stl file per component
- The body frame x-axis must be defined as the longitudinal axis of the vehicle, in the flow direction; the body frame z-axis must be defined pointing downwards
- When generating a mesh for the geometry, make sure all the facet normals in the .stl file point externally (outward of the body)
- Each component is assigned with a specific material, and assumed to be homogeneous
- The user can introduce additional materials and associated properties in the file DART_library/Material/database_material.xml, following the structure for the materials already provided
- if BLOOM is used, the volume mesh is hybrid with prismatic layers near the surface and tetrahedra elements in the remaining volume; otherwise, the volume mesh is fully unstructured;
- The Material name must be chosen from the material database in TITAN/Material/database_material.xml

CODE OUTPUTS

1. The simulation creates an output folder with the path given in the input option ***Output_folder***. Inside the output folder, there should be three folders:

    1.1 - Visualization – contains a list of .vtk files according to the frequency specified in the input option ***Output_freq.*** Each .vtk file provides a 3D solution of pressure (relative to freestream pressure), heat flux and temperature over the surface of the vehicle

    1.2 - Data – containing files:

        1.2.1 - data.csv

            a) data.csv contains output of each assembly for each time-step

            b) 1st column indicates the "Time" of the simulation, that is always incremented by the time-step

            c) 2nd column "Iter" shows associated iteration

            d) 3rd column "Assembly_ID" shows the ID of the assembly. Initially there is only have 1 assembly. After fragmentation the initial assembly is be split into 2 or more assemblies

            e) all quantities come in SI units excepts the angles FlightPathAngle,HeadingAngle,Latitude,Longitude,AngleAttack,AngleSideslip,Roll,Pitch,Yaw - which come in degrees

            f) the remaining columns show, for each assembly:

Mass,Altitude,Distance,Velocity,FlightPathAngle,HeadingAngle,Latitude,Longitude,AngleAttack,AngleSideslip,ECEF_X,ECEF_Y,ECEF_Z,ECEF_vU,ECEF_vV,ECEF_vW,BODY_COM_X,BODY_COM_Y,BODY_COM_Z,Aero_Fx_B,Aero_Fy_B,Aero_Fz_B,Aero_Mx_B,Aero_My_B,Aero_Mz_B,Lift,Drag,Crosswind,Inertia_xx,Inertia_xy,Inertia_xz,Inertia_yy,Inertia_yz,Inertia_zz,Roll,Pitch,Yaw,VelRoll,VelPitch,VelYaw,Quat_w,Quat_x,Quat_y,Quat_z,Quat_prev_w,Quat_prev_x,Quat_prev_y,Quat_prev_z,Mach,Speedsound,Density,Temperature,Pressure,SpecificHeatRatio,N2_mass_pct,O2_mass_pct,O_mass_pct,He_mass_pct,Ar_mass_pct,N_mass_pct,H_mass_pct,Pstag,Tstag,Rhostag,Aref,Lref

g) Distance: travelled distance of the assembly

h) ECEF_X,ECEF_Y,ECEF_Z,ECEF_vU,ECEF_vV,ECEF_vW: are the assembly COG position and velocity components in ECEF frame

i) BODY_COM_X,BODY_COM_Y,BODY_COM_Z: are the assembly COG position in the body frame

j) Aero_Fx_B,Aero_Fy_B,Aero_Fz_B,Aero_Mx_B,Aero_My_B,Aero_Mz_B: aerodynamic forces and moments in body frame

k) Quat_w,Quat_x,Quat_y,Quat_z,Quat_prev_w,Quat_prev_x,Quat_prev_y,Quat_prev_z: assembly current and previous quaternions (used to change quantities between ECEF and body frame)

l) Density,Temperature,Pressure,SpecificHeatRatio,N2_mass_pct,O2_mass_pct,O_mass_pct,He_mass_pct,Ar_mass_pct,N_mass_pct,H_mass_pct: atmospheric quantities for given altitude of the assembly

m) Pstag,Tstag,Rhostag: flow stagnation quantities

n) Aref,Lref: area and length of reference

1.2.2 - data_assembly.csv

a) contains detailed data of each assembly, i.e., contains data for each object of each assembly, essentially data for the individual objects

b) The columns are:

Time,Iter,Assembly_ID,Obj_name,Photons_second,Mass,Max_stress,Yield_stress,Parent_id

Parent_id: ID of the assembly to which object belongs

1.2.3 – the output files generated by the script postprocess_emissions.py are located in this folder and are named emissions_<iteration>.csv, with a file being generated for each iteration, according to the frequency requested in the input option **Frequency**, in input section **[Radiation]**

1.3 - Restart – contains solution restart files that can be used to restart the simulation. The restart files are essential to do post-processing for radiative emissions since the post-process is done on the basis of these solution files.

Running TITAN

1) running a trajectory simulation:

in the root TITAN folder, execute $ python TITAN.py -c <input>.txt

2) post-processing a solution

The solution can only be post-processed for iterations for which restart files are available, which will depend on user-defined options in the <input>.txt file. The restart files are located in <Output_folder>/Restart/

2.1) post-process 3D assembly output:

in the root TITAN folder, execute $ python TITAN.py -c input.txt -pp <tag>

Options for <tag>:

ECEF – output is generated in the ECEF frame
WIND – output is generated in with WIND frame

2.2) post-process radiative emissions:

in the root TITAN folder, execute $ python TITAN.py -c input.txt --emissions