FOSSEE, IIT Bombay
OpenFOAM Case Study
December 10, 2021

Pollutant Dispersion Modelling using CFD: A walkthrough of solver development in OpenFOAM

Binayak Lohani
Department of Mechanical and Aerospace Engineering
Pulchowk Campus,Nepal

Abstract

Alarming increase of the hazardous pollutants in the major South Asian cities such as Kathmandu, Delhi, Mumbai, etc risks the life of every individuals there. The situation worsens especially, in the winters rising the air quality index to life-risking situations. A proper scientific study and modelling of the pollutants is necessary in order to properly manage the pollution. A major source in the production of such harmful pollutants are from the vehicles and industries. A Computational Fluid Dynamics approach is proposed to model the pollutants using different turbulence models. The primary aim of the study is to develop a turbulent steady state solver for a passive transport of pollutants. The work is validated with the CEDVAL experiment which was conducted at Hamburg University. $k-\epsilon$ model better predicts the dispersion of the pollutant than $k-\omega$ SST turbulence model which over predicts the behaviour.

1 Introduction

Air pollution is one of the major challenges humankind is facing in the 21st century. Dispersion of pollutant is most due to the exhaust of vehicles and factories. Rapid urbanization of cities risks the life of many people due to increase in the air pollution. Urbanization also leads to the increase in the amount of traffic vehicles around the city area which impacts the air pollution. Air quality index (AQI) is represented by particulate matter (PM2.5 and PM10) i.e. particle with size less that 2.5mm and 10mm respectively. Cities in South Asian region has been facing an alarming increase in AQI along with China. Among 100 of the most polluted cities, 94 alone lies in India, China and Pakistan(Aljazeera (2021)).

Involvement of atmospheric flows and urban wind flow around building increases the complexity of the flow problem (Schatzmann et al. (2010)). A combination of solver that solves Navier Stokes equation with the passive scalar transport equation is appropriate to solve a pollutant dispersion problem (Yoshie et al. (2010)). Reynolds averaged Navier Stokes(RANS) turbulence closure model was used to strike a balance between accuracy and computational cost. Hence, $k-\epsilon$ and

$k-\omega SST$ models were compared against the experimental results at CEDVAL for a single isolated building. Previous studies include the use of Gaussian Plume model to predict the dispersion of pollutant from a particular source. However, this approach is unable to predict complex flow environments with turbulent phenomenon. An opensource software OpenFOAM-v2012 was employed for the development of the solver and simulating dispersion of the pollutant.

2 Problem Statement

Modelling of the emissions of the pollutant dispersion from any kind of sources as explained above requires a suitable model development in OpenFOAM. The pollutant such as CO_2,$N0_x$,etc was assumed as a scalar quantity which was being transported through air medium. The scalar acts as passive field which does not actively react or affect the wind flow from farfield. Hence, a passive scalar transport equation was incorporated into the steady state solver simpleFoam. Along with it, a turbulent Schmidt number was added as input which plays significant role in the diffusion of the passive scalar term.

A model which is used to validate the solver is shown in figure 1. An isolated single building with four emission sources assumed as pollutant from the garage is used.

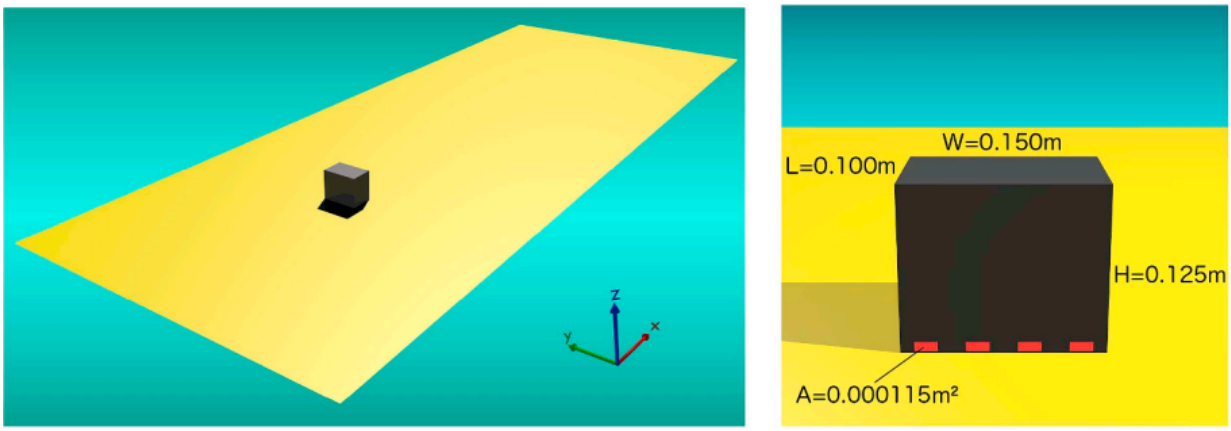


Figure 1: Geometry of CEDVAL A1-5 building (Longo et al. (2019))

Table 1: Dimensions of the configuration (in m)

	Length (x)	Width (y)	Height (z)
Building	0.1	0.15	0.125
Fluid domain	2.15	1	0.66

The non-dimensionalization of the co-ordinate axes was implemented as:

$$X = \frac{x}{H}, \qquad Y = \frac{y}{H}, \qquad Z = \frac{z}{H} \tag{1}$$

where, H = height of the building $(0.125m)$

3 Governing Equations and Models

Considering the incompressible effect of the fluids, Navier Stokes equation is discretized using the Finite Volume technique. The combination of mass and momentum equation contributes as NS equation whose mathematical formulation is given as (2) and (3) respectively.

$$\frac{\partial u_j}{\partial x_j} = 0 \tag{2}$$

$$\frac{\partial u_i}{\partial t} + \frac{\partial \left(u_i u_j\right)}{\partial x_j} = -\frac{\partial P}{\partial x_i} + \frac{1}{Re}\frac{\partial \left(\tau_{ij}\right)}{\partial x_j} \tag{3}$$

Since simpleFoam solver is a steady state solver, the temporal derivative term vanishes. However, time is represented with the iteration numbers and not in the real and physical sense. Now, scalarTransportFoam solver which is generally used for the transport of the passive scalar term has the equation as shown in equation 4.

$$\frac{\partial T}{\partial t} = \frac{\partial \left(D_{eff}\frac{\partial T}{\partial x_i}\right)}{\partial x_i} - \frac{\partial (u_i c)}{\partial x_i} + S_c \tag{4}$$

T represent the scalar parameter or concentration of the pollutant. D_{eff} is the diffusivity term which is the overall sum of molecular diffusivity (D) and eddy diffusivity (D_t). Eddy diffusivity is the ratio of turbulent viscosity to turbulent Schmidt number. The definition of total diffusivity is given in equation 5.

$$D_{eff} = D + \frac{\nu_t}{Sc_t} \tag{5}$$

where Sc_t=Schmidt number.

4 Solver Development

A combination of simpleFoam and scalarTransportFoam constitutes for the development of turbScalarTransportSimpleFoam. For the dispersion of the pollutant, a variable T denoting its concentration in ppm is represented in the solver. T acts as a passive scalar which is transported in a turbulent environment without being actively involved in the physics of the flow. Hence, passive in a sense that the chemical compound comprising the pollutant does not actively react with the ambient air.

The code turbScalarTransportSimpleFoam.C calls for the header files of each variable parameters such as U, p and T header files. These header file contains all the necessary equations and definitions for solving the problem. And, finally createFields.H is required to read all the necessary input parameters from the user in /constant/transportProperties folder. simpleFoam solver present in the /applications/solver/simpleFoam was customized in a way so that the scalar transport equation was entered in the respective manner as shown below:

4.1 turbScalarTransportSimpleFoam.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2011-2017 OpenFOAM Foundation
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

Application
    turbScalarTransportSimpleFoam

Description
    This is a combination of two solvers i.e. scalarTransportFoam and simpleFoam for t
    of the scalars. The code developed is used for a passive transport of the pollutan
     etc . Hence, the solver aims to model the pollutant dispersion phenomenon using C
     opensource software OpenFOAM.

\*---------------------------------------------------------------------------*/

#include "fvCFD.H"
#include "singlePhaseTransportModel.H"
#include "turbulentTransportModel.H"
#include "simpleControl.H"
#include "fvOptions.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[])
{
    argList::addNote
    (
        "Steady-state solver for incompressible, turbulent flows."
    );

    #include "postProcess.H"
    #include "addCheckCaseOptions.H"
    #include "setRootCaseLists.H"
    #include "createTime.H"
    #include "createMesh.H"
```

```
    #include "createControl.H"
    #include "createFields.H"
    #include "initContinuityErrs.H"

    turbulence->validate();

    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    Info<< "\nStarting time loop\n" << endl;

    while (simple.loop())
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;

        // --- Pressure-velocity SIMPLE corrector
        {
            #include "UEqn.H"
            #include "pEqn.H"
        }

        laminarTransport.correct();
        turbulence->correct();

        #include "TEqn.H"

        runTime.write();

        runTime.printExecutionTime(Info);
    }

    Info<< "End\n" << endl;

    return 0;
}


// ************************************************************************* //
```

4.2 UEqn.H

```
    // Momentum predictor

    MRF.correctBoundaryVelocity(U);

    tmp<fvVectorMatrix> tUEqn
```

```
(
    fvm::div(phi, U)
  + MRF.DDt(U)
  + turbulence->divDevReff(U)
 ==
    fvOptions(U)
);
fvVectorMatrix& UEqn = tUEqn.ref();

UEqn.relax();

fvOptions.constrain(UEqn);

if (simple.momentumPredictor())
{
    solve(UEqn == -fvc::grad(p));

    fvOptions.correct(U);
}
```

4.3 pEqn.H

```
{
volScalarField rAU(1.0/UEqn.A());
volVectorField HbyA(constrainHbyA(rAU*UEqn.H(), U, p));
surfaceScalarField phiHbyA("phiHbyA", fvc::flux(HbyA));
MRF.makeRelative(phiHbyA);
adjustPhi(phiHbyA, U, p);

tmp<volScalarField> rAtU(rAU);

if (simple.consistent())
{
    rAtU = 1.0/(1.0/rAU - UEqn.H1());
    phiHbyA +=
        fvc::interpolate(rAtU() - rAU)*fvc::snGrad(p)*mesh.magSf();
    HbyA -= (rAU - rAtU())*fvc::grad(p);
}

tUEqn.clear();

// Update the pressure BCs to ensure flux consistency
constrainPressure(p, U, phiHbyA, rAtU(), MRF);

// Non-orthogonal pressure corrector loop
```

```
    while (simple.correctNonOrthogonal())
    {
        fvScalarMatrix pEqn
        (
            fvm::laplacian(rAtU(), p) == fvc::div(phiHbyA)
        );

        pEqn.setReference(pRefCell, pRefValue);

        pEqn.solve();

        if (simple.finalNonOrthogonalIter())
        {
            phi = phiHbyA - pEqn.flux();
        }
    }

    #include "continuityErrs.H"

    // Explicitly relax pressure for momentum corrector
    p.relax();

    // Momentum corrector
    U = HbyA - rAtU()*fvc::grad(p);
    U.correctBoundaryConditions();
    fvOptions.correct(U);
}
```

4.4 TEqn.H

```
    {
volScalarField DTT ("DTT", DT + turbulence->nut()/Sct) ;

while (simple.correctNonOrthogonal())
        {
            fvScalarMatrix TEqn
            (
                fvm::ddt(T)
              + fvm::div(phi, T)
              - fvm::laplacian(DTT, T)
             ==
                fvOptions(T)
            );

            TEqn.relax();
```

```
            fvOptions.constrain(TEqn);
            TEqn.solve();
            fvOptions.correct(T);
        }
```

4.5 createFields.H

```
    {

Info<< "Reading field p\n" << endl;
volScalarField p
(
    IOobject
    (
        "p",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

Info<< "Reading field T\n" << endl;
volScalarField T
(
    IOobject
    (
        "T",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);


Info<< "Reading field U\n" << endl;
volVectorField U
(
    IOobject
    (
        "U",
        runTime.timeName(),
```

```
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

Info<< "Reading transportProperties\n" << endl;

IOdictionary transportProperties
(
    IOobject
    (
        "transportProperties",
        runTime.constant(),
        mesh,
        IOobject::MUST_READ_IF_MODIFIED,
        IOobject::NO_WRITE
    )
);

Info<< "Reading Schmidt number Sct\n" << endl;

dimensionedScalar Sct
(
    "Sct", dimless, transportProperties
);


Info<< "Reading diffusivity DT\n" << endl;

dimensionedScalar DT
(
    "DT", dimViscosity, transportProperties
);


#include "createPhi.H"


label pRefCell = 0;
scalar pRefValue = 0.0;
setRefCell(p, simple.dict(), pRefCell, pRefValue);
mesh.setFluxRequired(p.name());
```

```
singlePhaseTransportModel laminarTransport(U, phi);

autoPtr<incompressible::turbulenceModel> turbulence
(
    incompressible::turbulenceModel::New(U, phi, laminarTransport)
);

#include "createMRF.H"
#include "createFvOptions.H"
```

4.6 /system/controlDict/

```
    {

/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2006                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     turbScalarTransportSimpleFoam ;

startFrom       latestTime;

startTime       0;

stopAt          endTime;

endTime         2000;

deltaT          1;

writeControl    timeStep;
```

```
writeInterval   100;

purgeWrite      0;

writeFormat     binary;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;

// adjustTimeStep  yes;

// maxCo           1;


    functions
{
    tTransport
    {
      type            scalarTransport;
      libs            ("solverFunctionObjects");
      resetOnStartUp  no;
      field          T;
      schemesField    U;

      fvOptions
      {

        tSource
        {
         type        scalarFixedValueConstraint;
         enabled     true;

         scalarFixedValueConstraintCoeffs
         {
           selectionMode   cellSet;
            cellSet        selectedCells;
           volumeMode      specific;
```

```
            fieldValues
            {
              T             1;

            }
          }
        }
         unitySource
             {
                 type            scalarSemiImplicitSource;
                 enabled         true;

                 scalarSemiImplicitSourceCoeffs
                 {
                     selectionMode   cellSet;
                     cellSet        selectedCells;
                     volumeMode      specific;
                     injectionRateSuSp
                     {
                         T           (0.096 0);
                     }
                 }
             }


      }


    }

}

// ************************************************************************* //
```

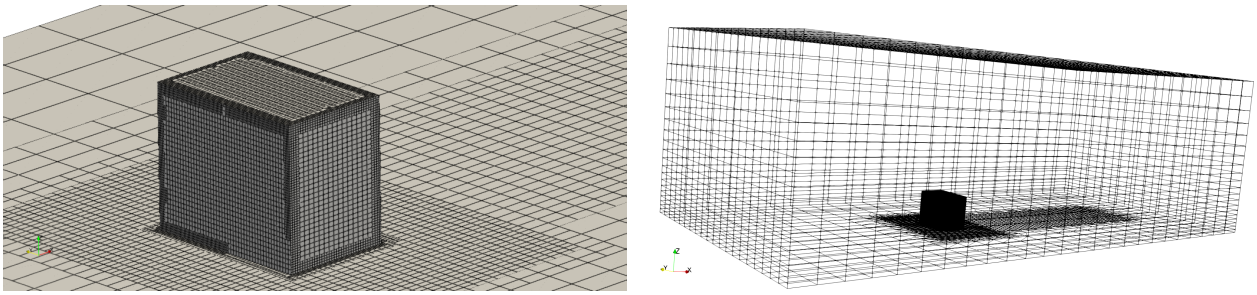
The source of pollutant assumed to be a garage emission at the bottom leeward side of the building is taken into account in /system/controlDict. fvOptions was used to define the source term of all four source element with 0.024g/s emission rate of each of them. Overall the rate was summed up to 0.096g/s. And, the concentration of the source term was defined as 1 ppm(part per million) in the same file.

5 Simulation Procedure

5.1 Geometry and Mesh

The domain consists of an isolated cuboid building. The fluid domain around the building is generated using blockMesh meshing utility of OpenFOAM. The dimension of the blockMesh geometry is given in table 1.

Next, the CAD model of the building was imported into the OpenFOAM folder. Further, grid generation was accomplished using snappyHexMesh. SnappyHexMesh is another inbuilt mesher of OpenFOAM which is very useful in constructing especially hexahedral cells in the periphery of the complex watertight geometry from CAD. Hence, the parameters used in snappyHexMesh dict are input so that good quality mesh are used during the simulation. y+ value 30 was used at the near walls of the building. 167,576 were the hexahedral mesh elements, 624 were prisms and finally, 14047 represent the polyhedral mesh elements constituting 182,249 total number of cells.



(a) (b)

Figure 2: Mesh configuration of the domain

5.2 Initial and Boundary Conditions

A suitable boundary condition is vital for the case study to depict the real world problem. A passive scalar quantity (T) is introduced in the 0 folder for initialization of the pollutant.

5.2.1 Inflow boundary condition

Depicting a real world scenario, we require a parabolic nature of the inlet velocity at the farfield. Atmospheric boundary layer concept was introduced so that the inlet velocity profile matches with the experimental condition conducted at lab. atmBoundaryLayerInletVelocity is an inbuilt boundary condition in OpenFOAM-v2012 that follows the same concept as discussed above. For the initialization of this particular condition, following parameters needs an input data:

U* is the friction velocity. Z0 is the roughness length Uref is the reference velocity at given height Zref.

5.2.2 Numerical boundary condition

Furthermore, other boundary condition which requires to be initialized at the beginning is given a detailed information in table 2-8

Table 2: Boundary condition for U

Patch	Condition	Value (ms^{-1})
Inlet	atmBoundaryLayerInletVelocity	(0, 0, 0)
Outlet	pressureInletOutletVelocity	(0, 0, 0)
ground	noSlip	-
building	noSlip	-
frontAndBack	symmetry	-

Table 3: Boundary condition for p

Patch	Condition	Value (m^2s^{-2})
Inlet	zeroGradient	-
Outlet	totalPressure	uniform 0
ground	zeroGradient	-
building	zeroGradient	-
frontAndBack	symmetry	-

Table 4: Boundary condition for nut

Patch	Condition	Value (m^2s^{-1})
Inlet	calculated	uniform 0
Outlet	calculated	uniform 0
ground	nutkWallFunction	uniform 0
building	nutkWallFunctiont	uniform 0
frontAndBack	symmetry	-

Table 5: Boundary condition for k

Patch	Condition	Value (m^2s^{-1})
Inlet	atmBoundaryLayerInletK	uniform 0
Outlet	inletOutlet	uniform 0.4
ground	kqRWallFunction	-
building	kqRWallFunction	-
frontAndBack	symmetry	-

Table 6: Boundary condition for omega

Patch	Condition	Value
Inlet	atmBoundaryLayerInletOmega	uniform 0
Outlet	inletOutlet	uniform 1.78
ground	omegaWallFunction	-
building	omegaWallFunction	-
frontAndBack	symmetry	-

Table 7: Boundary condition for epsilon

Patch	Condition	Value
Inlet	atmBoundaryLayerInletEpsilon	uniform 0
Outlet	inletOutlet	uniform 0.064
ground	epsilonWallFunction	-
building	epsilonWallFunction	-
frontAndBack	symmetry	-

Table 8: Boundary condition for T

Patch	Condition	Value
Inlet	fixedValue	uniform 0
Outlet	zeroGradient	-
ground	zeroGradient	-
building	zeroGradient	-
frontAndBack	symmetry	-

5.3 Solver

5.3.1 Numerical Solvers

Discretization of the NS equation into Finite Volume method requires a reliable solver for the effective and accurate problem solving. Along with it, the pressure-velocity coupling is another challenging aspect to be dealt with for the incompressible flows. Therefore, the solvers used for solving the Navier Stokes equation into the domain of interest are explained in table (9). Since, a steady state case was solved for the validation of the case study forward time marching scheme or solver need not to be defined.

Table 9: Numerical Solvers

Field	Linear Solver	Smoother	Tolerance
U	Smooth Solvers	Gauss Seidel Smoother	1e-06
p	GAMG Solver	Gauss Seidel Smoother	1e-05
nut	Smooth Solvers	Gauss Seidel Smoother	1e-06
k	Smooth Solvers	Gauss Seidel Smoother	1e-06
omega	Smooth Solvers	Gauss Seidel Smoother	1e-06
T	Smooth Solvers	Gauss Seidel Smoother	1e-06

6 Results and Discussions

6.1 Mesh Sensitivity Analysis

Optimal number of mesh elements required to accurately and efficiently solving a given problem is an important aspect. Three different mesh configurations were constructed and the results were therefore, compared among them. M1, M2 and M3 are the number of cells generated using snappyHexMesh. The case with M2 mesh configuration was approached for further analysis.

Table 10: Mesh Sensitivity Analysis

Mesh configuration	Number of cells
M1	28,197
M2	182,249
M3	323,176

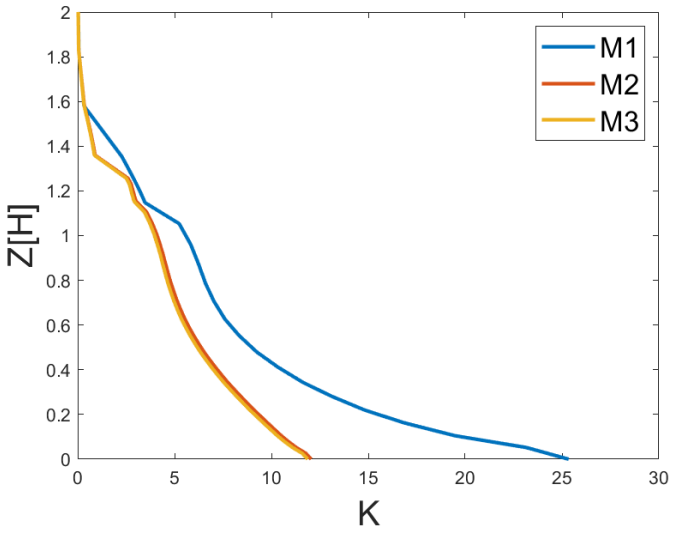


Figure 3: Mesh sensitivity Analysis

6.2 Validation of Atmospheric Boundary Layer boundary condition with the experiment

Implementation of atmospheric boundary layer boundary condition, scales the complexity of the problem. As explained above in section 5.2.1, the value of the required parameters set as boundary condition in OpenFOAM is shown in table 11.Therefore, with the given configuration, the inlet velocity profile is plotted against the height of the building. It can be concluded that the implemented boundary condition is validated with the experimental result from figure 4.

Table 11: Atmospheric Boundary Layer input parameters

Parameters	Value
U_{ref}	5.6
Z_{ref}	0.66
$z0$	0

6.3 Comparison between experimental and computational results

The case was run for 2000 iterations with residuals equivalent to 1e-04. However, due to no significant changes in the results, the case was not iterated furthermore for full convergence. Experiment conducted at Compilation of Experimental Data for Validation of Microscale Dispersion Models (CEDVAL) at Hamburg University was employed for the validation of the case study. A1-5 case using dispersion around a rectangular building is considered a standard for the verification of the dispersion model being developed. The non-dimensional number (K) was used to define the characteristics of the pollutant in the flow environment given as:

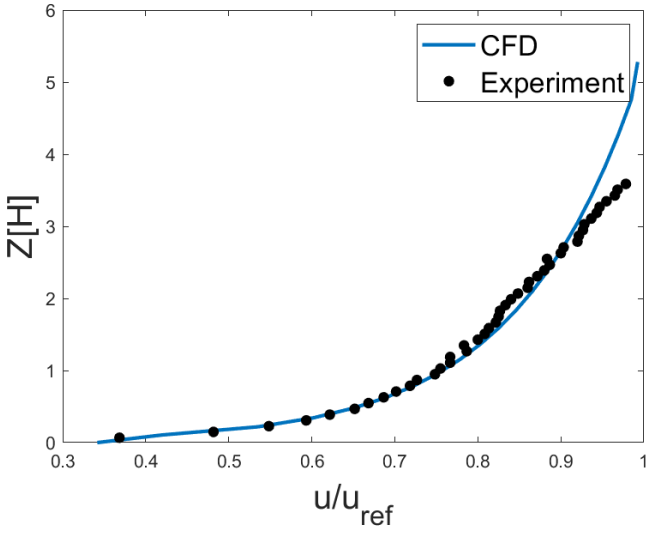
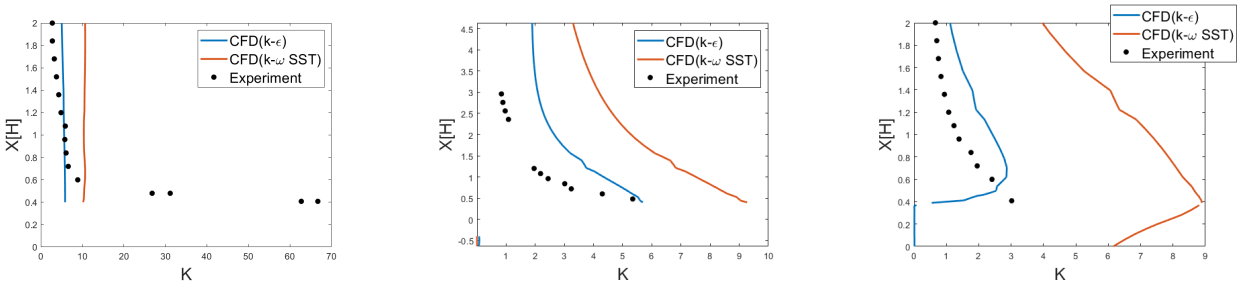


Figure 4: Velocity Profile of Atmospheric boundary layer

$$K = \frac{\frac{T_m}{T_s} U_{ref} H}{Q_s} \tag{6}$$

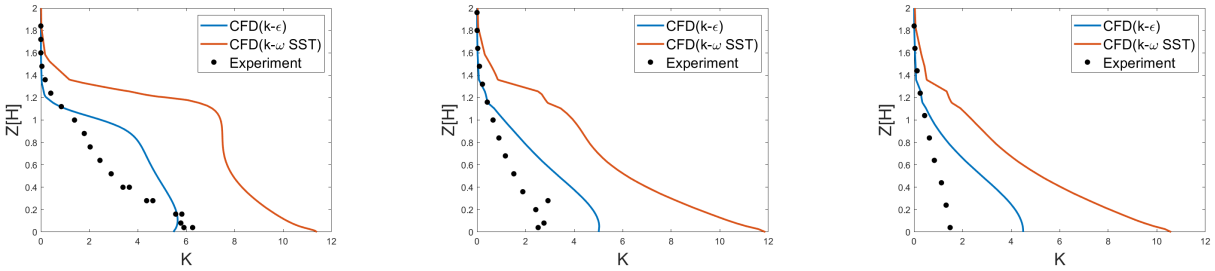
where,
T_m=measured concentration in ppm
T_s=source concentration in ppm
U_{ref}=reference velocity at given height
H=Height of the building
Q_s=Source flow rate in m^3/s

The graph is plotted along longitudinal(x) and vertical(z) direction at various different locations. It was seen that $k - \epsilon$ predicted closely to the experimental result than $k - \omega SST$ model which over-estimates the parameter K in different location of the setting.



(a) $Z = 0.08$ (b) $Z = 0.64$ (c) $Z = 1$

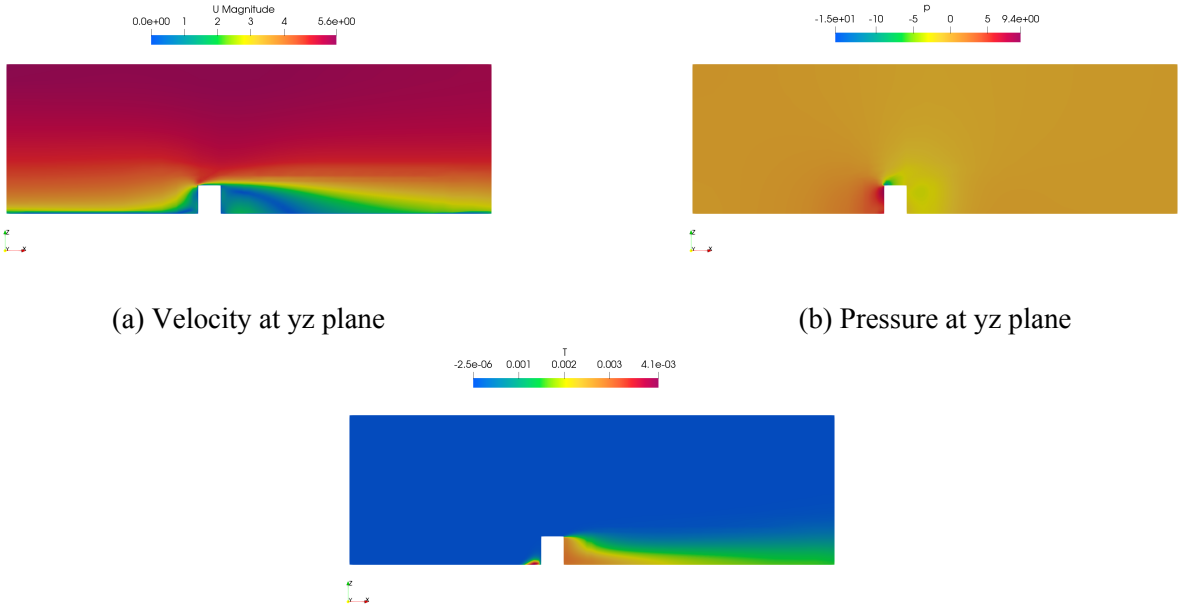
Figure 5: Comparision of K along X



(a) $X = 0.96$ (b) $X = 2$ (c) $X = 2.96$

Figure 6: Comparision of K along Z

6.4 Contours of flow parameters



(a) Velocity at yz plane

(b) Pressure at yz plane

(c) Concentration at yz plane

Figure 7: Contours of velocity, pressure and concentration (T) respectively

7 Conclusion and Future Works

The simulation performed for two different RANS turbulence closure models address that $k-\epsilon$ model closely validate with the experimental wind tunnel result than $k-\omega$ SST model. Concentration of the pollutants seem to diffuse less in the vertical direction and convects more towards the urban wind. A classical approach of using Gaussian plume model for modelling the pollutant dispersion cannot solve the flow problem with complex urban environment. Hence, CFD may play important role so that any urban environment can be accurately modelled using sophisticated

(a) Concentration at xy plane at Z=0.16 (b) Concentration at xy plane at Z=1

Figure 8: Contours of concentration (T) at xy plane

RANS model. However, LES and DNS approaches can also be used considering the computational cost and required degree of accuracy. Nonetheless, RANS approach is proven to be appropriate to strike the balance between both parameters.

Further, the work can be developed to solve complex urban environment of a particular city. Along with it, study on the dependance of the turbulent Schmidt number in different flow settings is the direction path a researcher might follow up on the future. An appropriate modelling of dispersion of pollutant helps in the prediction and management of the air pollution.

References

Aljazeera (2021). Air quality index. `(https://www.aljazeera.com/news/2021/11/22/infographic-the-worlds-100-most-polluted-cities-interactive`.

Longo, R., Fürst, M., Bellemans, A., Ferrarotti, M., Derudi, M., and Parente, A. (2019). Cfd dispersion study based on a variable schmidt formulation for flows around different configurations of ground-mounted buildings. *Building and Environment*, 154.

Schatzmann, M., Olesen, H., and Franke, J. (2010). *COST 732 model evaluation case studies: approach and results*.

Yoshie, R., ABE, S., and IIZUKA (2010). The fifth international symposium on computational wind engineering. *Wind Engineers, JAWE*, 35:347–363.