Computational Fluid Dynamics MSc

Home assignment II.
Wake of an Ahmed body

Introduction

A dream of many engineers is to be a vehicle designer. A few of them start to design vehicles in one of the university racing teams. First you have to learn about the limits of vehicle designing, before you start working on your own design using CFD.

The Ahmed body is a frequently studied simplified vehicle model. The primary motivation of the simulation of an Ahmed body is to evaluate the capability of the simulation tool to model the main flow structures around vehicles. Many versions of this body is introduced in the literature, from a simple bluff body to more complex geometries with tyres. For this simulation, the basic Ahmed body is considered.

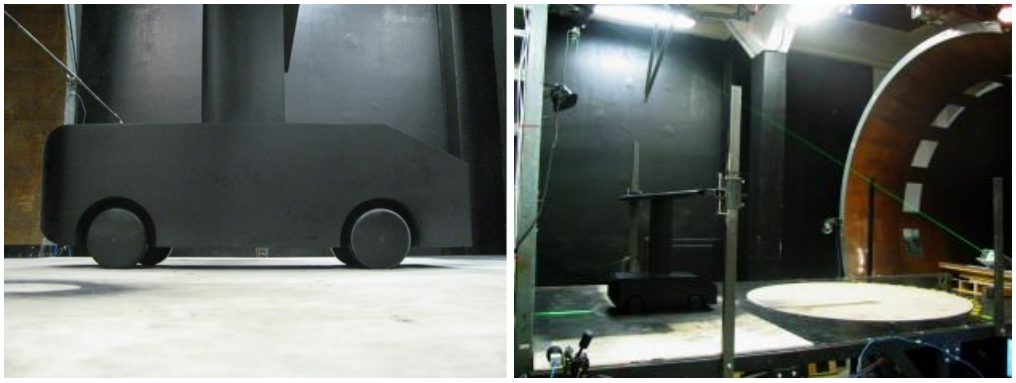


Figure 1. – Measurement of an Ahmed body with wheels at BME, AE building, Large wind tunnel [1]

Task

Simulate the flow field around a simplified car model (Ahmed Body) with a slant angle of 25° or 35°, mainly concentrating on the wake of the vehicle. For validation, Flow Around a Simplified Car Body (Ahmed Body) can be used (http://cfd.mace.manchester.ac.uk/ercoftac/doku.php?id=cases:case082).

- Make a 3D half model (symmetry) according to the validation case.
- Run the steady-state simulation with 3 different turbulence models (use 2-equations turbulence models in two cases and at least one more complex model in one case).
- Compare the available pressure and velocity data.
- Give suggestions for the preferable turbulence model, based on their ability to reproduce the wake
- Write a technical report about your simulations.

Documentation

In the documentation, you should write down your investigation in your OWN word. On the diagrams and figures, your OWN result should be presented. The document and the calculation need to be uploaded onto Moodle.

The documentation should contain the following parts:

- Aim of the simulation
- Introduction
    - Introduce the measurement
        - What was measured, where did it take place and by which equipment
        - Uncertainty of the measured variables
    - Refer to a few papers that made a simulation for this case
        - What type of mesh they used
        - What kind of (turbulence) model they used
- Describe the geometry (with figures)
- Describe the mesh (with figures)
- Describe the physical models
- Results of the simulation with different turbulence model
    - velocity, pressure contour plots
    - y+ values
    - compare velocity and pressure plots with the measurement results
- Summary
- References

Data for validation

At least the following data should be used for the validation:

- Pressure distribution:
    - centerline (y = 80)
- LDA measurement:
    - x-z Planes: y = 0, 100, 180
    - y-z Planes: x = -88, 0, 80, 200

References

[1] András Gulyás, Ágnes Bodor, Tamas Regert, Imre M. Jánosi (2013*)- PIV measurement of the flow past a generic car body with wheels at LES applicable Reynolds number*, International Journal of Heat and Fluid Flow, Volume 43, Pages 220-232, ISSN 0142-727X, https://doi.org/10.1016/j.ijheatfluidflow.2013.05.012.

Budapest University of Technology and Economics
Faculty of Mechanical Engineering

Computational Fluid Dynamics
II. Homework

Gergő Kelle

December 3, 2023

Contents

1 Introduction

1.1 Aim of the simulation

The aim of the simulation is to investigate the flow around a simplified car body known as the *Ahmed body*. Furthermore it's aim is to explore the impact of the slant angle (25° or 35°) on three-dimensional flow patterns.

1.2 Reference experiment

The reference experiment conducted a detailed study of the *Ahmed body* within the same wind tunnel, focusing on slant angles (25° and 35°) for the rear section. During the experiments they used advanced measurement techniques, including hot-wire and two-component LDA systems, to provide benchmark data on mean velocity, Reynolds stress profiles, and surface pressure coefficients for aerodynamic analyses.

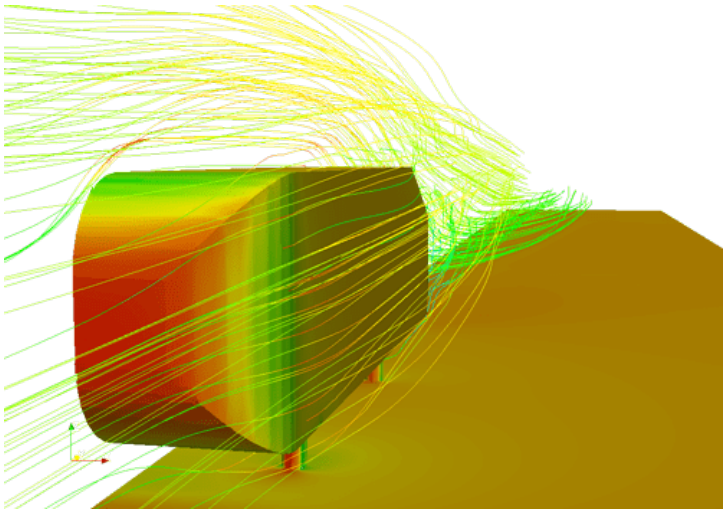


Figure 1: Illustration of the *Ahmed body* simulation

Table 1: Summary of Reference Experiment Data

Parameter	**Value**	**Units**
Cross Section of Wind Tunnel	1.87×1.4	m^2
Stilt Diameter	30	mm
Ground Clearance	50	mm
Slant Angles	25, 35	°
Slanting Section Length	222	mm
Kinematic Viscosity of Air	15×10^{-6}	m^2/s
Bulk Velocity	40	m/s
Body Height	288	mm
Reynolds Number	768,000	1
Blockage Ratio	4	%

2 Pre-processing

2.1 Geometry

The geometry was created based on Figure 2, with an addition that I've choosed the slant angle to be 35° as Figure 3 shows.

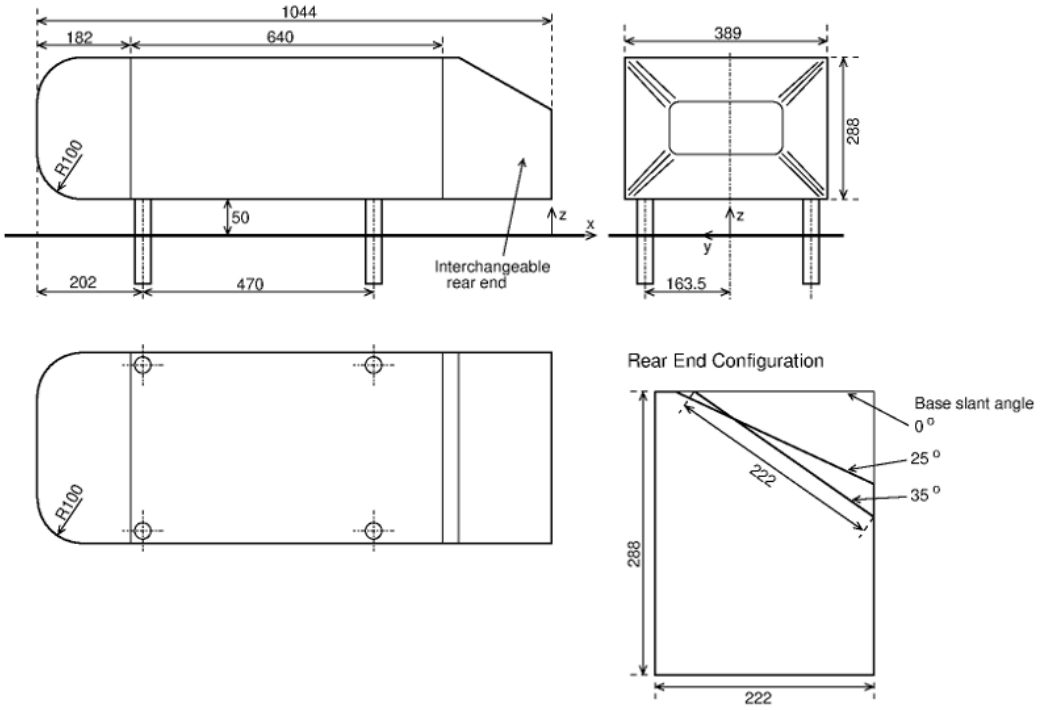
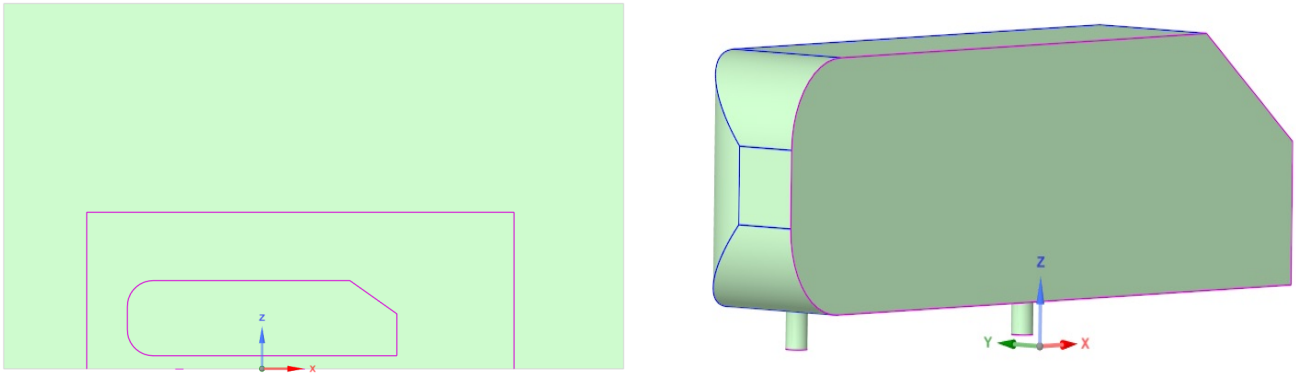


Figure 2: *Ahmed body* geometry [[2]]



(a) Close and far field

(b) Ahmed body

Figure 3: The geometry used for simulaton

2.2 Discretization - Meshing

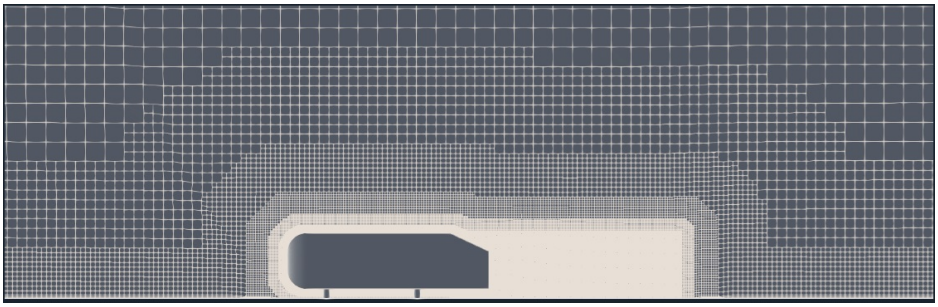


Figure 4: Example for meshing for *Ahmed body* simulation [1]

Based on the literature the main similarities between each meshing is that they always have a close field, and a far field with different sizing, while applying inflation around the body. Thus I've applied sizing, and inflation meshing methods to create the mesh which is showed in Figure 5.

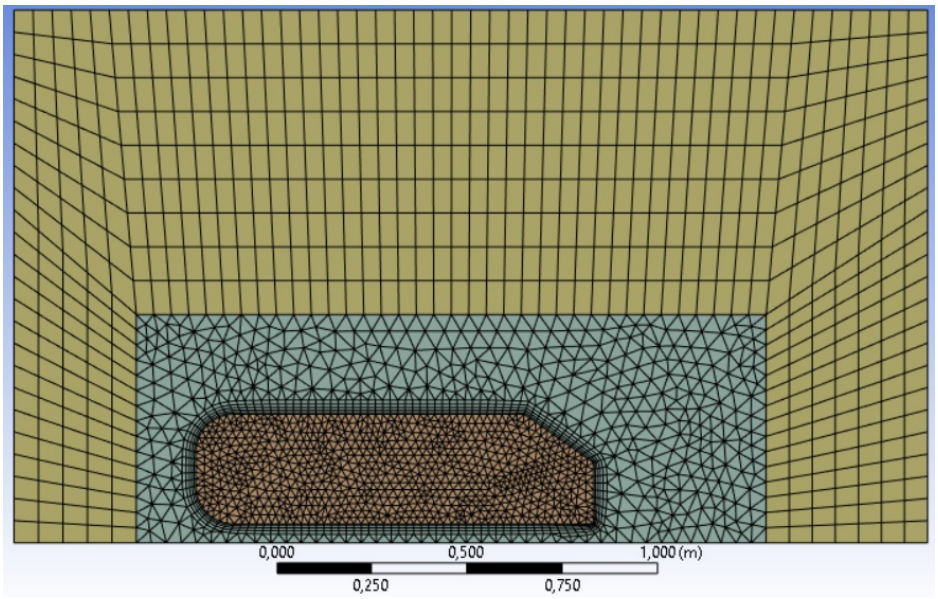


Figure 5: The mesh used for simulaton

Table 2: Mesh Statistics

Nodes	Elements	Min. element quality (worst)
346007	207823	0.066

2.3 Defining the Properties of the Model

As for the simulation properties I've chosed two 2-equations turbulence and one more complex model. The ones that I've choosed is present in Table 3.

Table 3: Summary of Reference Experiment Data

Model	Option type	Option
$k-\varepsilon$	Model	Standard
$k-\omega$	Near-wall treatment	Standard wall function
Transition SST	Option	Production Kato-Launder & Limiter

Before the turbulence model I've assigned name slections in order to have proper boundary conditions such as *velocity inlet*, *pressure outlet*, *symmetry* for the side and top walls, and *wall* name selection for the ground and the body, while assigning *fluid* name selection for both the close and far field.

As for the calculation settings I've turned convergence criteria to none, and set the iteration number for the simulation to 800.

Since I had to do the calculation for 3 different turbulence model I've created a simulation model tree to ensure that I use the same model and meshing showed in Figure 6.

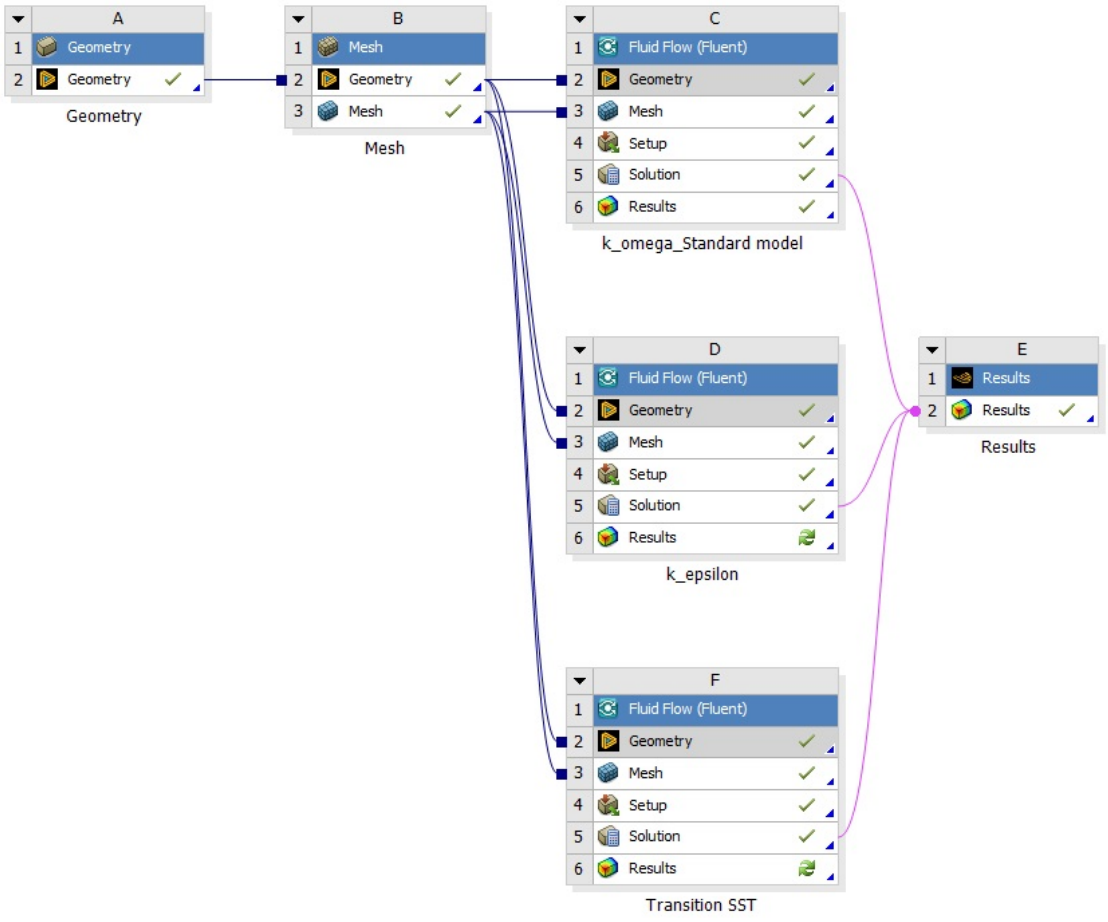


Figure 6: The mesh used for simulaton

3 Post-processing

> *i: Part of the post-processing were done using Python. The used code is available as appendix.*

3.1 Drag coefficient

Speaking of drag coefficient the $k-\omega$ and the $k-\varepsilon$ gave a very similar solution while *Transitions SST* gave a noticably different solution as Figure 7 shows.

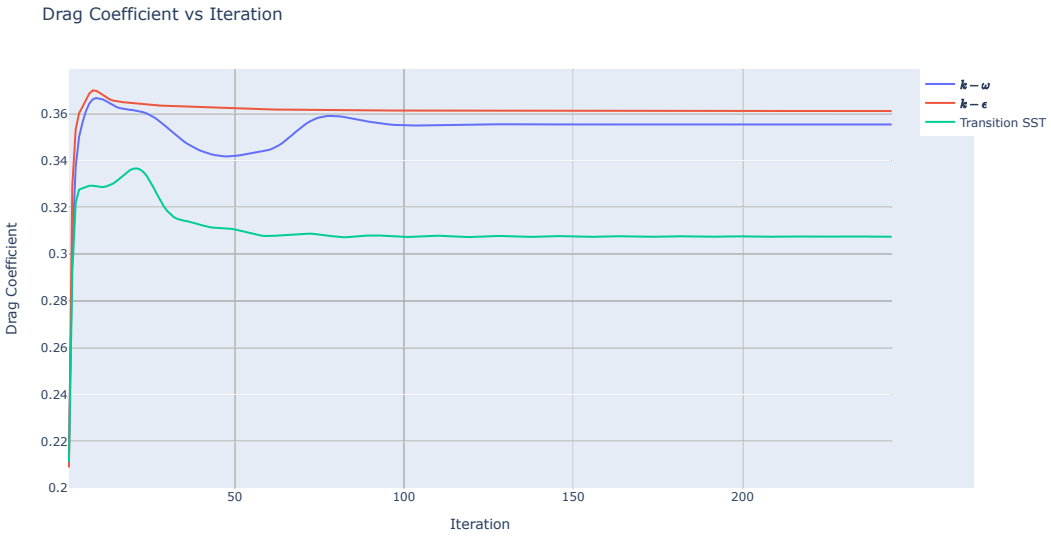


Figure 7: Drag coefficient results for 3 type of turbulent models

The *Transitions SST* turbulence model is generally considered more accurate for simulating flows with transitional boundary layers, as it combines elements of both the $k-\omega$ and $k-\varepsilon$ models. In the case of Ahmed body simulation, where complex flow separation and reattachment occur, the *Transitions SST* model can better capture the transitional nature of the boundary layer, providing a more accurate representation of the flow phenomena compared to the standard $k-\omega$ or $k-\varepsilon$ models.

On the other hand it is important to point out that in case of drag coefficient *Transitions SST* turbulence model gave almost the same solution as if the $k-\omega$ turbulence model were used with SST near-wall treatment instead of standard.

Table 4: Drag Coefficient values for Different Turbulence Models

Model	**Drag coefficient**
$k-\varepsilon$	0.361
$k-\omega$	0.355
Transition SST	0.307

3.2 Velocity and Pressure

For faster post-processing I've connected the results of each simulation into a single result viewer, thus the following figures about the simulation will have the $k-\omega$ at the top-left, the $k-\varepsilon$ at the top-right and the *Transition SST* at the bottom!

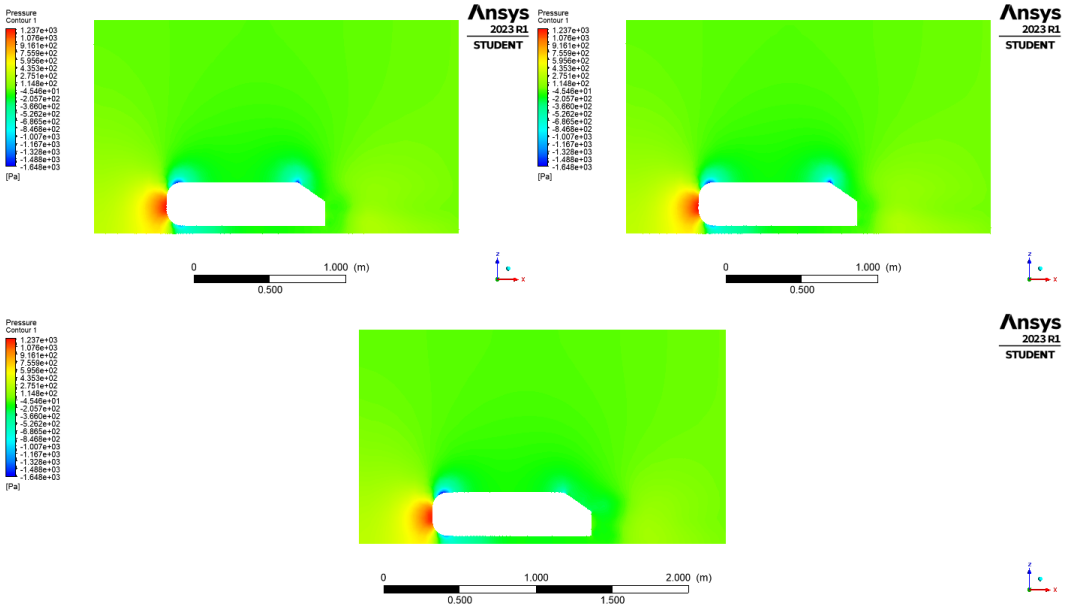


Figure 8: Pressure Contour Plot for Different Turbulence Models

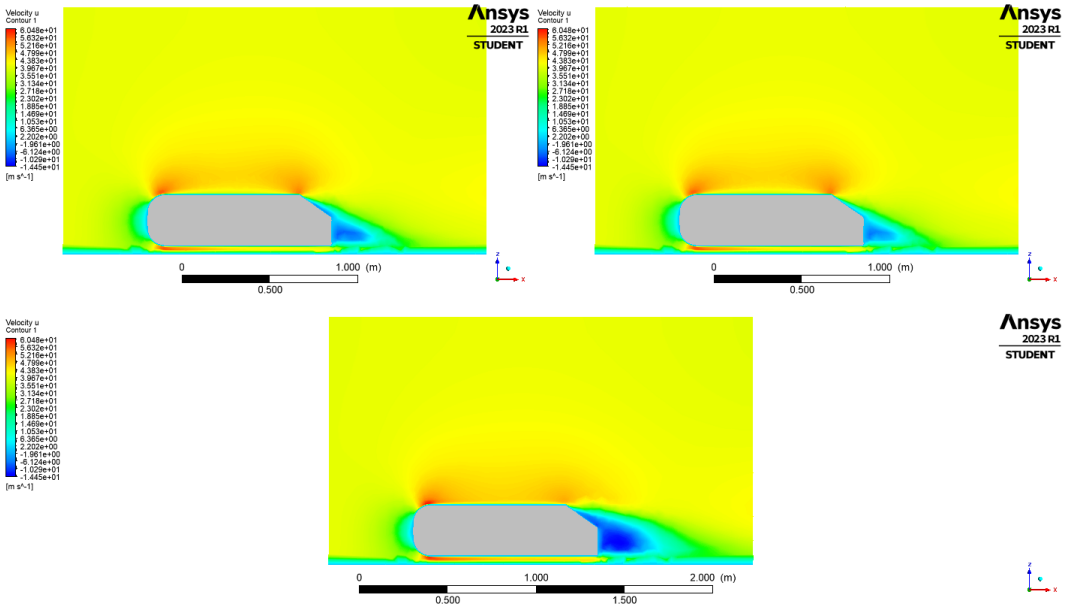
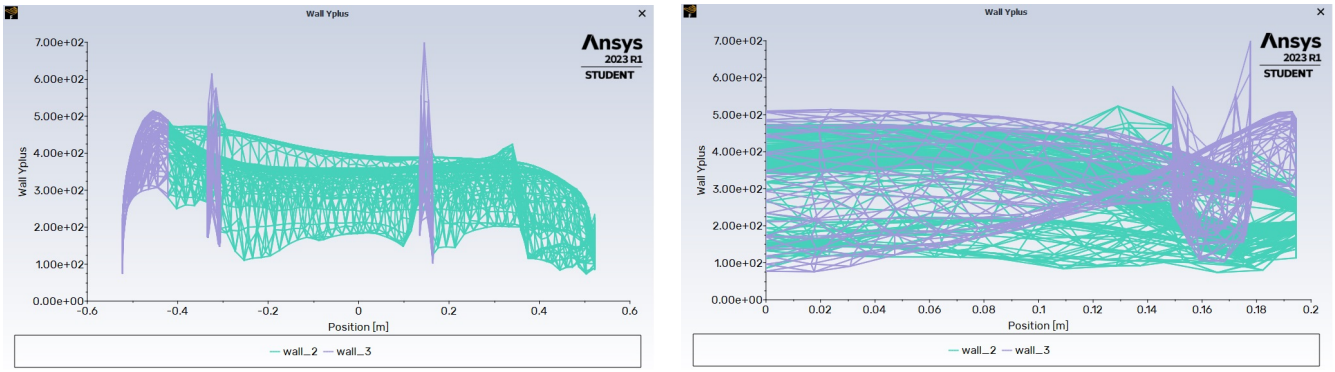


Figure 9: Velocity (u) Contour Plot for Different Turbulence Models

3.3 y^+ value

In CFD, y^+ is a dimensionless parameter representing the distance of the first computational grid point from the wall normalized by the molecular viscous length scale. It is essential for accurate modeling of near-wall turbulent flows, with low y^+ values indicating proper grid resolution within the viscous sublayer.

The y^+ values for the surface of the *Ahmed body* is visualized in Figures 10a and 10b where the purple lines stands for the stilts on which the body was mounted meanwhile the green lines represent the *Ahmed body* it self.



(a) From [1 0 0]

(b) From [0 1 0]

Figure 10: y^+ Distributions on Different Walls

In CFD simulations, a y^+ value in the range of 30 to 300 is often considered appropriate. The observed outcome, spanning a numerical range between 80 and 600. This result signifies that the inflation applied during mesh generation has led to the development of a rather coarse mesh in proximity to the boundary layer.

3.4 Comparison

A good comparison viewpoint could be the simulated and the measured drag coefficient value.

Table 5: Summary of Reference Experiment Data

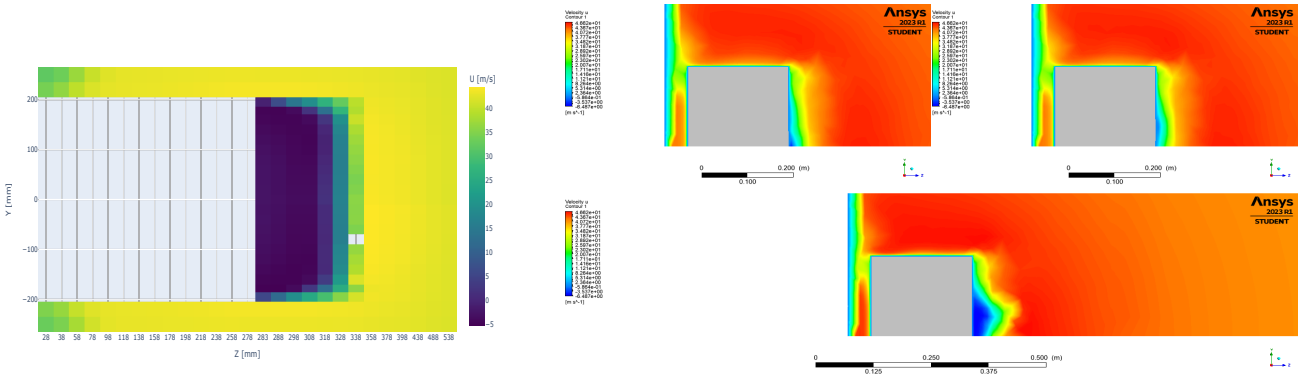
Model	Drag coefficient	Difference [%]
$k-\varepsilon$	0.361	20.3
$k-\omega$	0.355	18.3
Transition SST	0.307	2.3
Measured [3]	0.300	Base

Based on the Table 5 results, the Transition SST turbulence model gave the most accurate results to the measured drag coefficient.
Furthermore for data validation the homework assignment asked to validate our results on certain planes based on the LDA measurements.

Table 6: LDA Measurement Data Planes

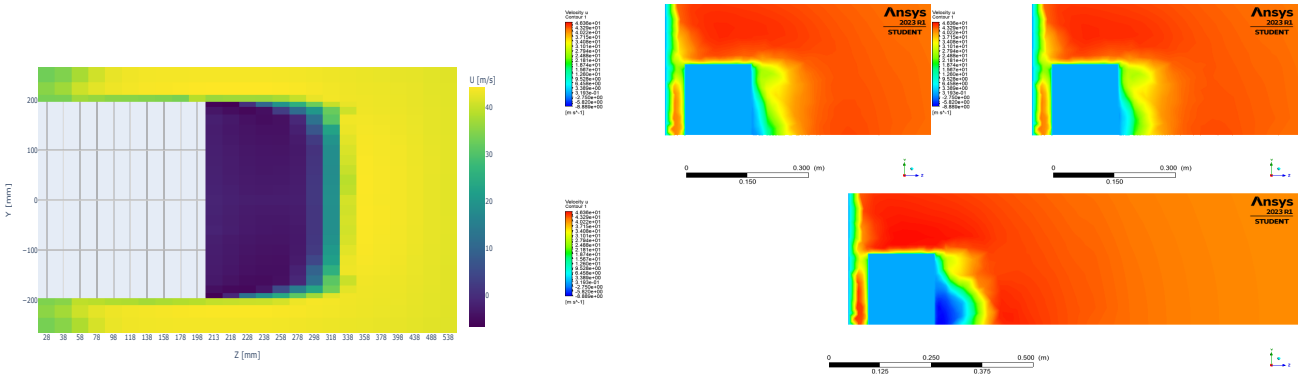
Plane	Parameters
y-z	$x = -88$
y-z	$x = 0$
y-z	$x = 80$
y-z	$x = 200$
x-z	$y = 0$
x-z	$y = 100$
x-z	$y = 180$

Keep in mind that in case of measured data a full *Ahmed body* was measured while we just simulated the half of it due to symmetry.



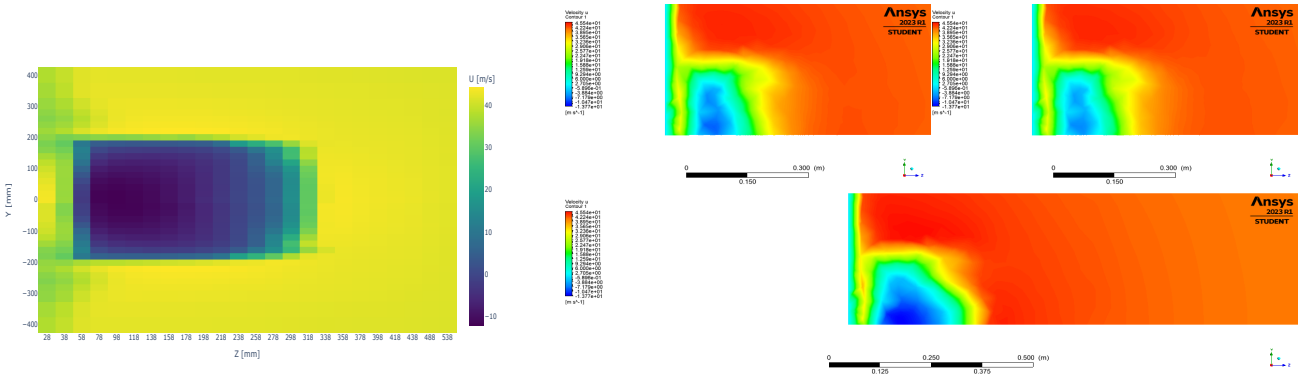
(a) Measured data visualized (b) Simulation results

Figure 11: Velocity (u) in the Y-Z Plane ($x = -88$)



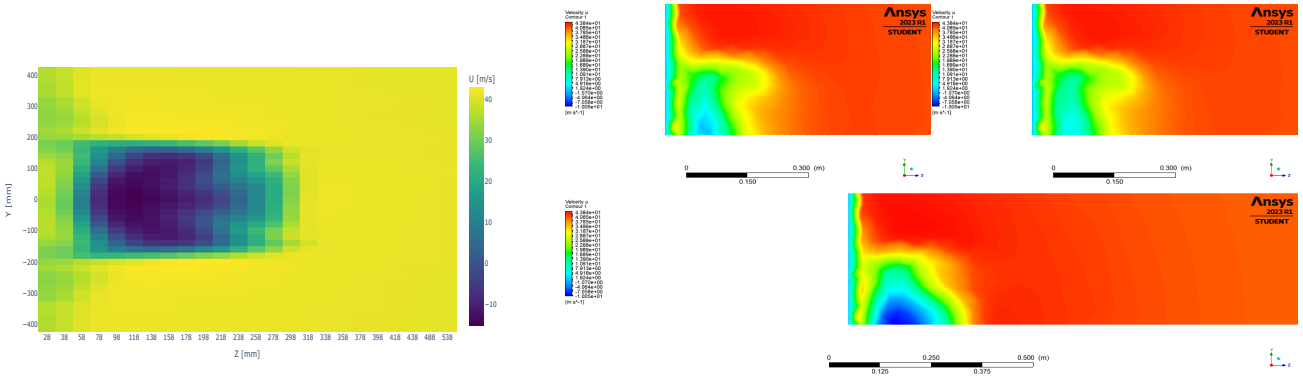
(a) Measured data visualized (b) Simulation results

Figure 12: Velocity (u) in the Y-Z Plane ($x = 0$)



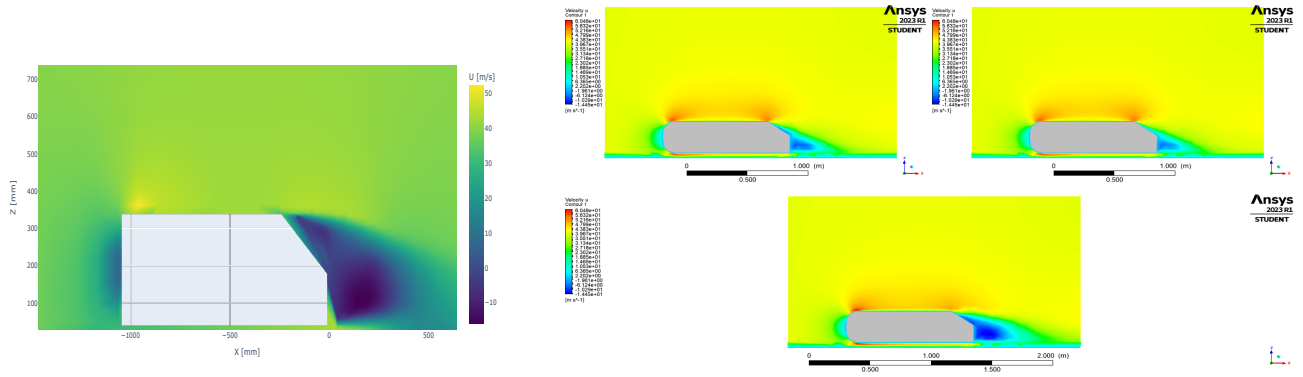
(a) Measured data visualized (b) Simulation results

Figure 13: Velocity (u) in the Y-Z Plane ($x = 80$)



(a) Measured data visualized (b) Simulation results

Figure 14: Velocity (u) in the Y-Z Plane ($x = 200$)



(a) Measured data visualized (b) Simulation results

Figure 15: Velocity (u) in the X-Z Plane ($y = 0$)

Speaking of the results based on Figure 11 - 15 shows similarities between the measured data of [1] and my simulation results. Thus ensuring the validation of the results.

4 Summary

As for a summarization in this homework I've compared 3D steady state simulations with 3 different turbulence modells including two 2 equation modells ($k-\omega$ and $k-\varepsilon$) and one more complicated modell (Transition SST). The homework showed that not just the turbulent model it self but the near-wall treatment also have a high impact on the results. The most accurate turbulence model out of theese three is appears to be the Transition SST based on the drag coefficient comparison.

The comparison showed similarities in flow characteristics between the visualized measured data and the simulation results. On the other hand the difference between the simulation results and the measured data's drag coefficient is rather high, thus a denser mesh would be needed (based on Figure 10).

References

[1] Lienhart, H., Becker, S., Stoots, C. *Flow Around a Simplified Car Body (Ahmed Body)*. Experiments by H. Lienhart, S. Becker, C. Stoots. http://cfd.mace.manchester.ac.uk/ercoftac/doku.php?id=cases:case082.

[2] *Ahmed Body CFD Simulation Benchmark*. This study shows a complex CFD analysis and validation of external aerodynamics benchmark Ahmed body using simulation environment TCAE. https://www.cfdsupport.com/ahmed-body-tcfd-benchmark.html.

[3] Nakashima, T., Mutsuda, H., Kanehira, T., Tsubokura, M. *Fluid-Dynamic Force Measurement of Ahmed Model in Steady-State Cornering*. https://www.mdpi.com/1996-1073/13/24/6592.

5 Appendix

CFD_2HW_Code

December 3, 2023

[24]:
```
import plotly.graph_objects as go
import pandas as pd
import plotly.io as pio

# Read the data from the first file
file_path_1 = r'C:
 ↪\Users\kelle\HW2\hw2_cfd_gbbnul_files\dp0\FFF\Fluent\drag_coeff-rfile_formed.
 ↪txt'
df1 = pd.read_csv(file_path_1, delim_whitespace=True, skiprows=1,␣
 ↪names=['Iteration', 'drag_coeff'])

# Read the data from the second file
file_path_2 = r'C:
 ↪\Users\kelle\HW2\hw2_cfd_gbbnul_files\dp0\FFF-1\Fluent\drag_coeff-rfile_formed.
 ↪txt'
df2 = pd.read_csv(file_path_2, delim_whitespace=True, skiprows=1,␣
 ↪names=['Iteration', 'drag_coeff'])

# Read the data from the third file
file_path_3 = r'C:
 ↪\Users\kelle\HW2\hw2_cfd_gbbnul_files\dp0\FFF-2\Fluent\drag_coeff-rfile_formed.
 ↪txt'
df3 = pd.read_csv(file_path_3, delim_whitespace=True, skiprows=1,␣
 ↪names=['Iteration', 'drag_coeff'])

# Find the common range of iterations
common_range = range(max(df1['Iteration'].min(), df2['Iteration'].min(),␣
 ↪df3['Iteration'].min()), min(df1['Iteration'].max(), df2['Iteration'].max(),␣
 ↪df3['Iteration'].max()) + 1)

# Interpolate missing values for the first dataset
df1 = df1.set_index('Iteration').reindex(common_range).interpolate().
 ↪reset_index()

# Interpolate missing values for the second dataset
df2 = df2.set_index('Iteration').reindex(common_range).interpolate().
 ↪reset_index()
```

```
# Interpolate missing values for the third dataset
df3 = df3.set_index('Iteration').reindex(common_range).interpolate().
 ↪reset_index()

# Create traces for each dataset
trace1 = go.Scatter(x=df1['Iteration'], y=df1['drag_coeff'], mode='lines',␣
 ↪name='$k-\omega$')
trace2 = go.Scatter(x=df2['Iteration'], y=df2['drag_coeff'], mode='lines',␣
 ↪name='$k-\epsilon$')
trace3 = go.Scatter(x=df3['Iteration'], y=df3['drag_coeff'], mode='lines',␣
 ↪name='Transition SST')

# Plot the data using Plotly
fig = go.Figure(data=[trace1, trace2, trace3])
fig.update_layout(title='Drag Coefficient vs Iteration',␣
 ↪xaxis_title='Iteration', yaxis_title='Drag Coefficient')

# Save the plot as an HTML file
html_file_path = 'drag_coeff_plot.html'
pio.write_html(fig, html_file_path)

# Save the plot as a PDF file
pdf_file_path = 'drag_coeff_plot.pdf'
pio.write_image(fig, pdf_file_path, format="pdf", width=1600/1.5, height=900/1.
 ↪5, scale=1.5)

# Show the plot (optional)
fig.show()

# Calculate the mean value for the last one-third of the first dataset
one_third_index_1 = int(len(df1) * 2 / 3)
last_one_third_mean_1 = df1['drag_coeff'].iloc[one_third_index_1:].mean()

# Calculate the mean value for the last one-third of the second dataset
one_third_index_2 = int(len(df2) * 2 / 3)
last_one_third_mean_2 = df2['drag_coeff'].iloc[one_third_index_2:].mean()

# Calculate the mean value for the last one-third of the third dataset
one_third_index_3 = int(len(df3) * 2 / 3)
last_one_third_mean_3 = df3['drag_coeff'].iloc[one_third_index_3:].mean()

print(f'Mean value for the last one-third of Dataset 1:␣
 ↪{last_one_third_mean_1}')
print(f'Mean value for the last one-third of Dataset 2:␣
 ↪{last_one_third_mean_2}')
```

```
print(f'Mean value for the last one-third of Dataset 3:␣
 ↪{last_one_third_mean_3}')
```

```
Mean value for the last one-third of Dataset 1: 0.3554033817073171
Mean value for the last one-third of Dataset 2: 0.3611688097560976
Mean value for the last one-third of Dataset 3: 0.3075041658536585
```

[25]:
```
print("Error value")
kom = (last_one_third_mean_1 / 0.306 - 1) *100
print(f"k - omega   = {kom:.4f} %")
keps = (last_one_third_mean_2 / 0.306 - 1) *100
print(f"k - epsilon = {keps:.4f} %")
TSST = (last_one_third_mean_3 / 0.306 - 1) *100
print(f"TSST        = {TSST:.4f} %")
```

```
Error value
k - omega   = 16.1449 %
k - epsilon = 18.0290 %
TSST        = 0.4916 %
```

[31]:
```
import pandas as pd
import plotly.graph_objects as go
import numpy as np
from scipy.interpolate import griddata

# Read the data from the file
file_path = r'C:\Users\kelle\HW2\hw2_cfd_gbbnul_files\dp0\FFF\Fluent\asd.txt'
df = pd.read_csv(file_path, delim_whitespace=True, names=['x', 'y', 'z', 'U'])

# Convert 'x', 'y', and 'U' columns to numeric values
df[['x', 'y', 'z', 'U']] = df[['x', 'y', 'z', 'U']].apply(pd.to_numeric,␣
 ↪errors='coerce')

# Set a tolerance for y=0 filtering
tolerance_up = 1
tolerance_down = -1

# Filter data where y is close to 0
df_y_0 = df[(abs(df['y']) > tolerance_down) & (abs(df['y']) < tolerance_up)]

# Check if 'x' values are numeric
if not pd.api.types.is_numeric_dtype(df_y_0['x']):
    raise ValueError("'x' values are not numeric.")

# Check if 'z' values are numeric
if not pd.api.types.is_numeric_dtype(df_y_0['z']):
    raise ValueError("'z' values are not numeric.")
```

```
# Create a 1000x1000 grid
x_vals = np.linspace(df_y_0['x'].min(), df_y_0['x'].max(), 1000)
z_vals = np.linspace(df_y_0['z'].min(), df_y_0['z'].max(), 1000)

X, Z = np.meshgrid(x_vals, z_vals)

# Create a mask for the rectangular domain to exclude from interpolation
mask_rectangular_domain = ((X >= -1044) & (X <= -243) & (Z >= 40) & (Z <= 338))
mask_rectangular_domain2 = ((X >= -1044) & (X <= 0) & (Z >= 40) & (Z <= 178))

# Create a mask for the parallelogram-shaped domain to exclude from␣
 ↪interpolation
def is_point_inside_parallelogram(x, z):
    v0 = np.array([-243, 50])
    v1 = np.array([0, 178])
    v2 = np.array([-243, 338])
    v = np.column_stack((x.flatten(), z.flatten()))

    d1 = np.cross(v1 - v0, v - v0)
    d2 = np.cross(v2 - v1, v - v1)
    d3 = np.cross(v0 - v2, v - v2)

    return (d1 >= 0) & (d2 >= 0) & (d3 >= 0)

mask_parallelogram_domain = is_point_inside_parallelogram(X, Z)

# Combine both masks
combined_mask = mask_rectangular_domain | np.reshape(mask_parallelogram_domain,␣
 ↪mask_rectangular_domain.shape) | mask_rectangular_domain2

# Interpolate U values onto the new grid
points = np.column_stack((df_y_0['x'], df_y_0['z']))
U_interp = griddata(points, df_y_0['U'], (X, Z), method='linear')

# Set values in the specified domains to NaN
U_interp[combined_mask] = np.nan

# Create a heatmap using Plotly
fig = go.Figure()

heatmap = go.Heatmap(
    x=x_vals,
    y=z_vals,
    z=U_interp,
    colorscale='viridis',  # You can change the colorscale as needed
    colorbar=dict(title='U [m/s]'),
)
```

```
fig.add_trace(heatmap)

# Set labels for axes
fig.update_layout(
    #title='Heatmap of Y=0 ',
    xaxis=dict(title='X [mm]'),
    yaxis=dict(title='Z [mm]')
)

# Show the plot

# Save the plot as an HTML file
html_file_path = 'y100.html'
pio.write_html(fig, html_file_path)

# Save the plot as a PDF file
pdf_file_path = 'y100.pdf'
pio.write_image(fig, pdf_file_path, format="pdf", width=1600/1.5, height=900/1.
↪5, scale=1.5)

# Show the plot (optional)
fig.show()
```

[30]:
```
import pandas as pd
import plotly.graph_objects as go

# Read the data from the file
file_path = r'C:\Users\kelle\HW2\hw2_cfd_gbbnul_files\dp0\FFF\Fluent\asd.txt' ␣
↪# Replace with the actual file path
df = pd.read_csv(file_path, delim_whitespace=True, names=['x', 'y', 'z', 'U'])

# Convert 'x' column to numeric values
df['x'] = pd.to_numeric(df['x'], errors='coerce')

# Convert 'y' column to numeric values
df['y'] = pd.to_numeric(df['y'], errors='coerce')

# Set a tolerance for x=-88 filtering
x_target = 200
tolerance_x = 1

# Filter data for the YZ plane at x=-88 with tolerance
df_x_minus_88 = df[(df['x'] > (x_target - tolerance_x)) & (df['x'] < (x_target␣
↪+ tolerance_x))]

# Create a heatmap using Plotly for U in the YZ plane at x=-88
```

```
fig = go.Figure()

heatmap = go.Heatmap(
    x=df_x_minus_88['z'],
    y=df_x_minus_88['y'],
    z=df_x_minus_88['U'],
    colorscale='viridis',  # You can change the colorscale as needed
    colorbar=dict(title='U [m/s]')
)

fig.add_trace(heatmap)

# Set labels for axes
fig.update_layout(
#    title='Heatmap of U in the YZ plane at X=-88',
    xaxis=dict(title='Z [mm]'),
    yaxis=dict(title='Y [mm]')
)

# Show the plot

# Save the plot as an HTML file
html_file_path = 'x200.html'
pio.write_html(fig, html_file_path)

# Save the plot as a PDF file
pdf_file_path = 'x200.pdf'
pio.write_image(fig, pdf_file_path, format="pdf", width=1600/1.5, height=900/1.
↪5, scale=1.5)

# Show the plot
fig.show()
```