Experimental validation of the T4 Mach 7.0 nozzle.

School of Mechanical & Mining Engineering
Research Report Number 2014/14
(Also UQ HIFiRE8 Technical Report Number 2014/01)

W. Y. K. Chan, M. K. Smart and P. A. Jacobs
Centre for Hypersonics, The University of Queensland

September 2014

Abstract

This report details the experimental validation of an axisymmetric Mach 7.0 contoured nozzle that has recently been designed for the T4 reflected shock tunnel facility. This report details the experimental validation of the design of this nozzle. The validation is conducted by comparing Pitot pressure distributions that are experimentally measured at the exit of the nozzle with Pitot pressure distributions that are expected from the design process. An excellent agreement between the experimental and numerical Pitot pressure distributions is achieved at all three locations downstream of the nozzle exit, hence demonstrating the validity of the methodology that was used to design the nozzle flowpath. The experimentally-measured Pitot pressure distribution showed that the nozzle outflow is reasonably axisymmetric and that there is good level of uniformity in the core flow region of the nozzle flowfield. Additional tests indicated that the distribution of Pitot-to-nozzle-supply pressure ratio and the size of the uniform region of the nozzle flowfield are not sensitive to changes in the nozzle-supply conditions.

Contents

Nomenclature

Roman

H	Enthalpy
M	Mach number
p	Pressure
r	Radius
Re	Reynolds number
T	Temperature
x	Axial distance from nozzle-exit
y, z	Cartesian coordinates
y^+	Non-dimensionalised distance between the cell centre and and the wall

Subscript

0	Total conditions
Pitot	Pitot
s	Nozzle-supply or total conditions
u	Unit

1 Introduction

An axisymmetric contoured nozzle tailored for specific flight conditions of $M = 7$, $p_0 = 6.035\,\mathrm{MPa}$, $T_0 = 2432.2\,\mathrm{K}$ has recently been designed for the T4 reflected shock tunnel facility (Chan et al., 2013). This report details the experimental validation of the design of this nozzle. The validation is conducted by comparing Pitot pressure distributions that are experimentally measured at the exit of the nozzle with Pitot pressure distributions that are expected from the design process.

Section 2 describes the experimental setup and the test conditions that were used in this validation study, while Section 3 describes the CFD simulations that were conducted to support the analysis. The analysis of the experimental and numerical results is presented in Section 4.

2 Experimental setup

The survey of the nozzle outflow was conducted via a rake that is mounted with 33 Pitot probes. The Pitot rake is shown in Figure 1. Each of the Pitot probe is instrumented with

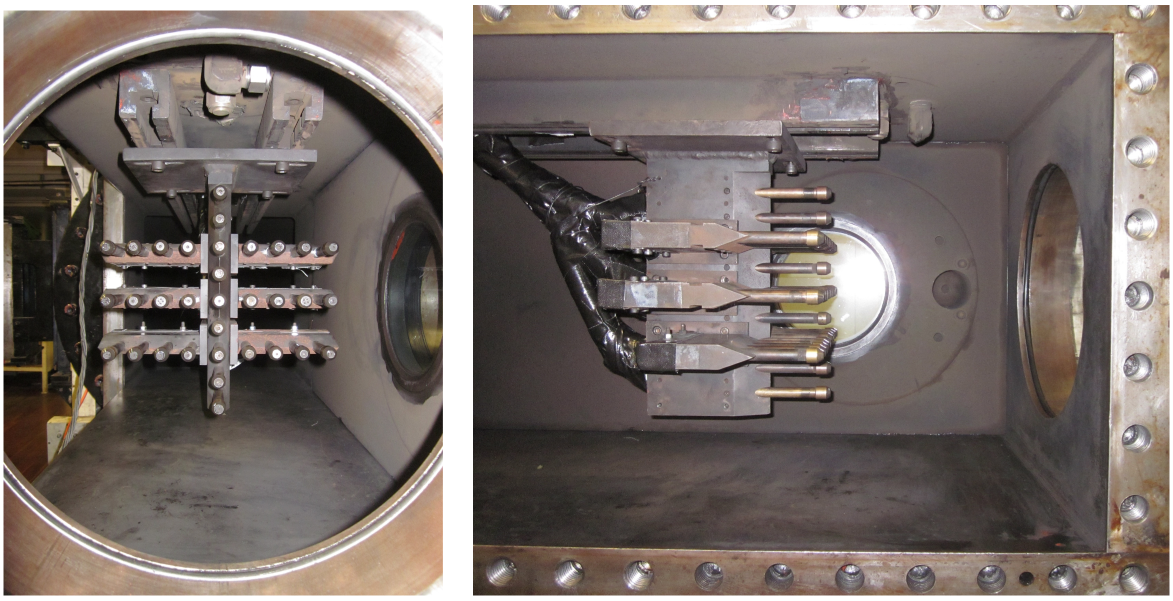

(a) Front view of Pitot rake. (b) Side view of Pitot rake.

Figure 1: Pitot rake in T4 test section.

a fast-response piezoelectric PCB pressure transducer, as shown in Figure 2. The PCB transducers were either of model number 112A21 or 112A22.The sensitivities of each PCB transducer were obtained from in-situ calibrations against a pre-calibrated Kulite XTEL-190-500A transducer (serial number LL5-41). Note also that the sensing face of each PCB transducer was shielded with a cellophane sheet to prevent effects of thermal shocking on the pressure measurements. Two of the 33 probes had Pitot head caps with swirl holes

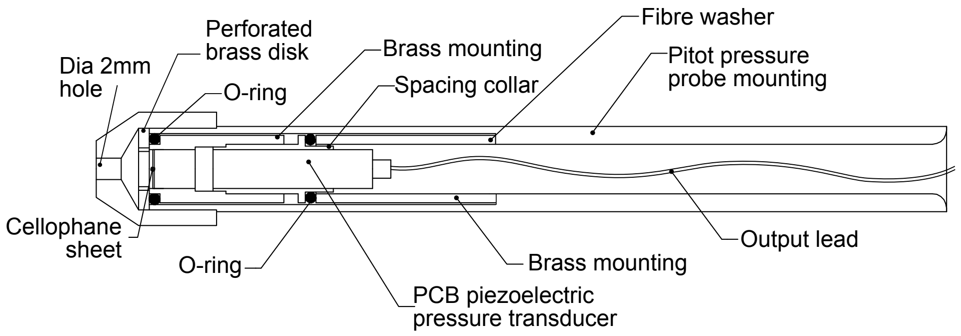
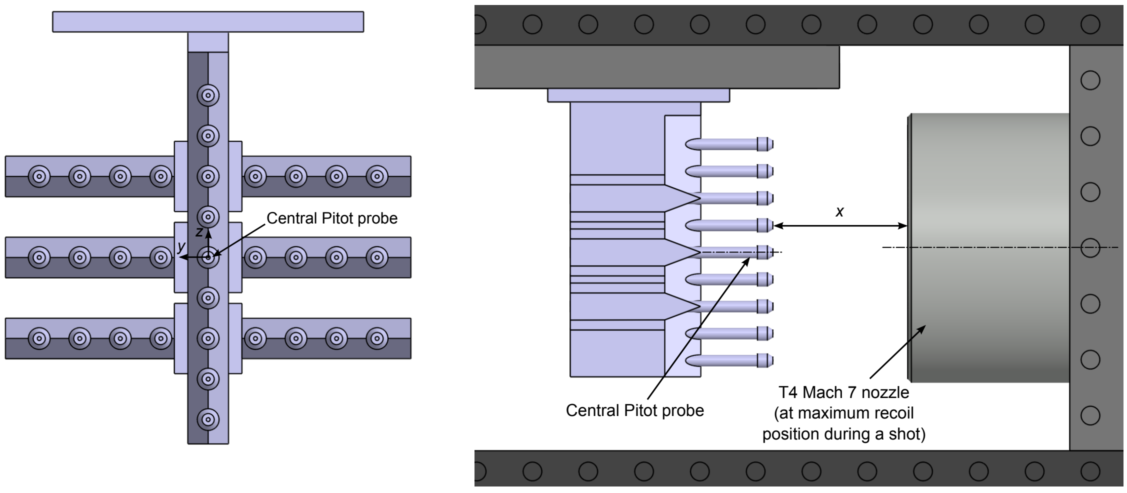


Figure 2: Schematic of each Pitot probe assembly (adapted from Smith (1999)).

rather than the 2 mm-diameter hole (that is shown on the front of the Pitot probe in Figure 2). The swirl holes were proposed by McGilvray et al. (2009) to reduce the noisy signals commonly seen in Pitot measurements. However, for the present experiments, no difference was observed in the noise levels in Pitot measurements from the swirl-hole head caps and from the conventional head caps.

A schematic of the Pitot rake is shown in Figure 3. During these experiments, surveys were conducted at three planes located axially downstream of the nozzle exit - $x = 141$ mm, $x = 301$ mm and $x = 460$ mm. At each plane, surveys were done with the rake in two positions - one with the rake shifted such that the central probe is located at $y = 0$ mm, and the other with the rake shifted such that the central probe $y = -12$ mm.



(a) Front view of Pitot rake.
(b) Side view of Pitot rake.

Figure 3: Schematic of Pitot rake in T4 test section.

Majority of the tests for the present study were conducted at a nominal total pressure of 19.33 MPa, total temperature of 2731 K and total enthalpy of 2.48 MJ/kg. This nominal test condition is the same as the test condition that is to be used for the ground-testing of the subscale HIFiRE 8 scramjet engine. Note though that the nozzle was designed for a total pressure of 6.035 MPa, total temperature of 2432 K and total enthalpy of 2.44 MJ/kg.

3 CFD simulation of nozzle and test section

As the nominal test conditions were different to those that the nozzle was designed for, a CFD simulation of the nozzle and test section had to be repeated with the appropriate inflow conditions[1]. The nozzle flowfield was first simulated using NENZFr (Doherty et al., 2012). The axisymmetric simulation was started at the sonic throat of the nozzle with inflow conditions at a total pressure of 19.33 MPa and total temperature of 2731 K. Exit flow profiles from the nozzle simulation were then used as inflow to an axisymmetric simulation of the test section flowfield. This simulation was conducted using Eilmer (Gollan & Jacobs, 2013). The setup script for the test section simulation can be found in Appendix B. Note that although this axisymmetric simulation does not model the true test section of the T4 shock tunnel (the actual test section has a square cross-section), it still does provide a reasonable estimate of the flowfield. The flow of air through the nozzle and test section was assumed to be in chemical and thermal equilibrium. The boundary layer was assumed to transition to turbulence 50 mm downstream of the nozzle throat[2]. The walls of the nozzle and test section were assumed to be at a constant temperature

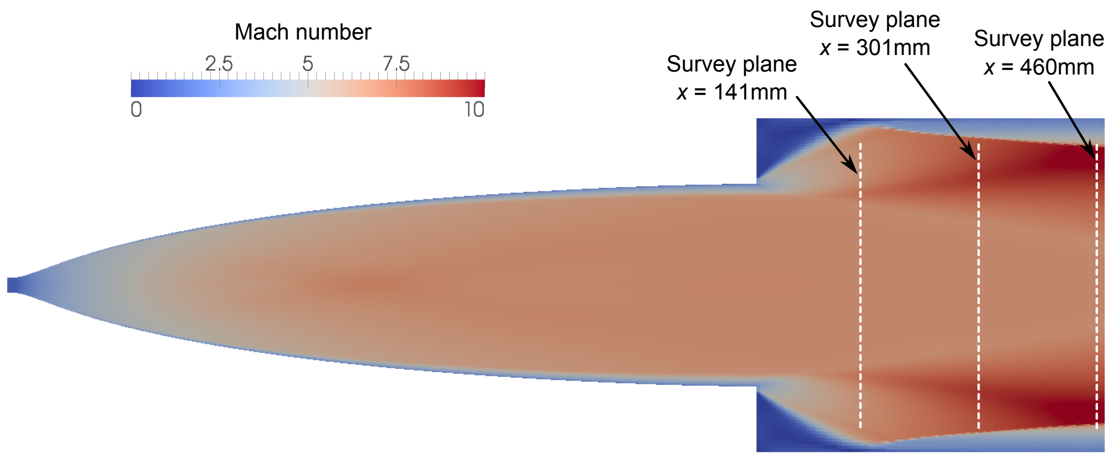


Figure 4: CFD solution showing iso-contours of Mach number.

of 300 K. Grid clustering was employed near the nozzle and test section walls to resolve the boundary layer. The y^+ values of most near-wall cells were under 1.0. Figure 4 shows

[1]The simulation was conducted using Revision 219 of NENZFr and Eilmer.

[2]Chan et al. (2013) showed that the nozzle outflow is not significantly altered when the boundary layer transitioned between 50 mm and 150 mm downstream of the throat.

the CFD solution with iso-contours of Mach number plotted. Also shown are the axial locations where the Pitot pressure distributions are surveyed for the experiments.

Note that although the actual T4 test section has a square cross-section, it was modelled in the present axisymmetric simulation to have a circular cross-section (with the diameter of the simulated test section matching the height of the actual test section). This was done to avoid having to do a three-dimensional simulation. Since the locations where the Pitot pressure distributions were measured are well upstream of where the expansion of the nozzle outflow reflects off the test section walls, this simplification is still expected to provide a reasonably good estimate of the Pitot pressure distribution in the core flow region of the nozzle outflow.

4 Results & discussions

Comparisons between the experimental and numerical Pitot pressure distributions are shown in the following subsections. For each comparison, the Pitot pressure distributions are presented in the form of a Pitot-to-nozzle-supply pressure ratio. For the experiments, this ratio is obtained by first shifting the nozzle-supply pressure trace later in time, such that the nozzle-supply pressure rises at the same time as the Pitot pressure. This is done to account for the time taken for the flow to arrive at the Pitot probe from the nozzle-supply region. Once the trace is shifted in time, the transient Pitot pressure trace is then normalised by the transient nozzle-supply pressure trace to give the transient Pitot-to-nozzle-supply pressure trace. A value for the Pitot-to-nozzle-supply pressure ratio is then calculated by averaging the transient Pitot-to-nozzle-supply pressure ratio values within the duration of the test window. The 1 ms test window is taken to start 2.2 ms after the test flow has arrived at the Pitot probe[3]. The uncertainty bars shown for the experimental results in the following sections are taken to be the standard deviation of the value of the averaged Pitot-to-nozzle-supply pressure ratio within the test window duration.

4.1 Results at nominal test condition

Figures 5, 6 and 7 show the comparisons between the experimental and numerical Pitot pressure distributions at three axial planes downstream of the nozzle exit. Note that the radial positions for the experimental data points on these plots are obtained by converting the y and z locations of each Pitot probe to a radial location based on a coordinate system which has its axis aligned with that of the nozzle. For a nozzle outflow that is approximately or truly axisymmetric, the application of this conversion method is appropriate. However, for a nozzle outflow that is not axisymmetric, this method of conversion will result in a large scatter in the Pitot pressure distribution. Figures 5,

[3]The start of the test window was selected based on that needed for the HIFiRE 8 ground testing. Typically, the test window can be taken to start approximately 0.8 - 1 ms after test flow arrival.

6 and 7 show that the scatter in the distribution of averaged Pitot pressure is small compared to the experimental uncertainties, hence indicating that the nozzle outflow is reasonably axisymmetric.

The plots in Figures 5, 6 and 7 show an excellent agreement between the experimental and numerical Pitot pressure distributions at all three locations downstream of the nozzle exit, hence indicating that the CFD solution is a very good estimate of the actual nozzle flowfield. The results also show an excellent level of uniformity in the core flow region of the nozzle flowfield. This, together with the excellent agreement between experiments and numerical simulations, demonstrates the validity of the methodology that was used to design the nozzle flowpath.

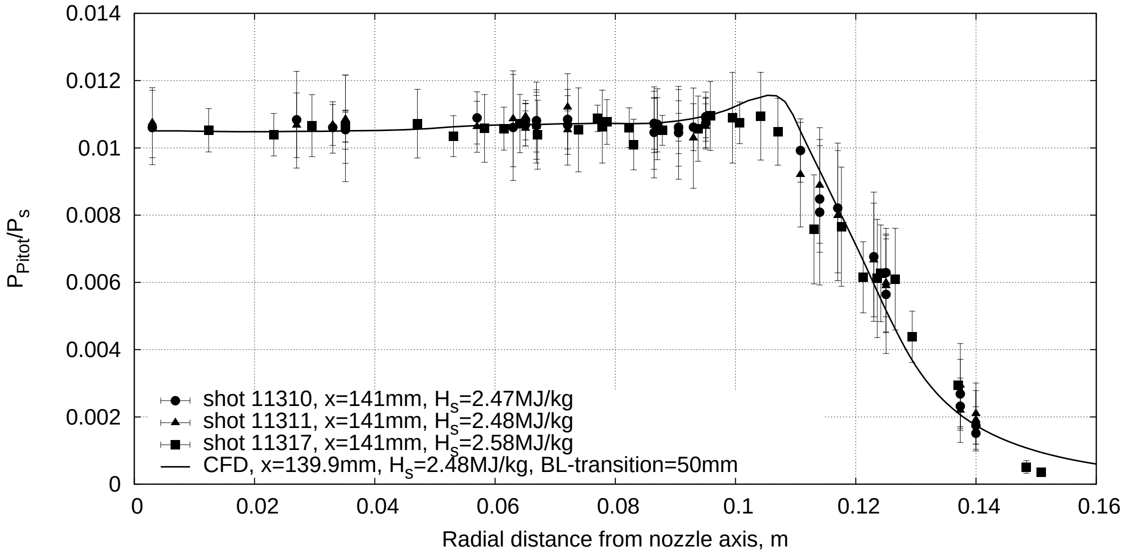


Figure 5: Pitot pressure distribution at $x = 141\,\text{mm}$.

4.2 Effect of test conditions on nozzle outflow

The nozzle was also surveyed at two other test conditions to investigate the sensitivity of the nozzle outflow to nozzle-supply conditions. The first test condition has a total pressure of 18.46 MPa, total temperature of 1707 K, total enthalpy of 1.60 MJ/kg and a unit Reynolds number of 9.8×10^6 /m, while the second test condition has a total pressure of 13.02 MPa, total temperature of 1735 K, total enthalpy of 1.63 MJ/kg and a unit Reynolds number of 6.7×10^6 /m. Figure 8 shows a comparison of the Pitot pressure distributions for the nominal test condition and for the other two test conditions. Note that the x positions for the tests at the other two conditions are slightly different to those

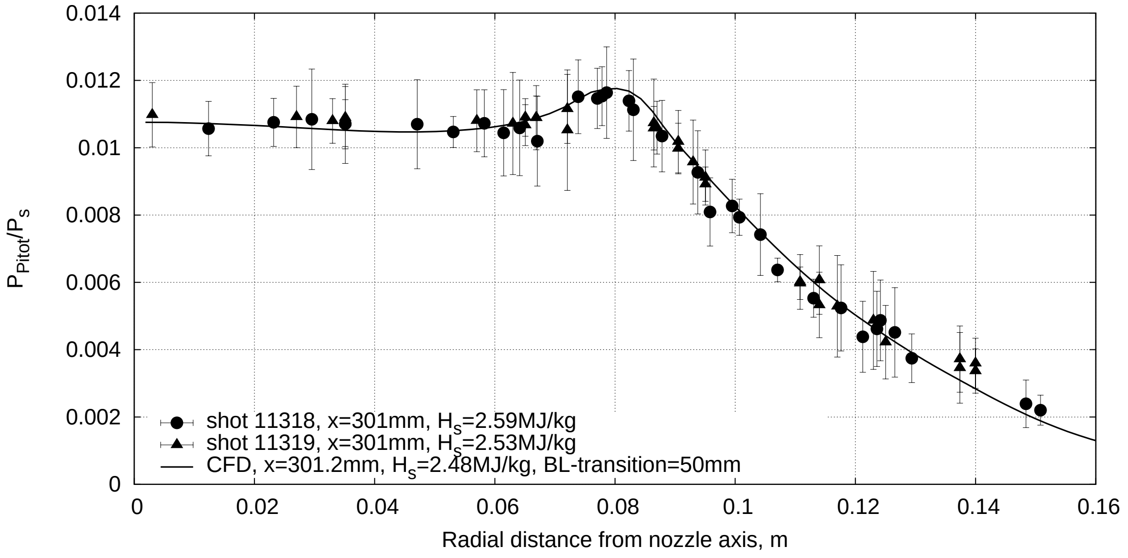


Figure 6: Pitot pressure distribution at $x = 300\,\text{mm}$.

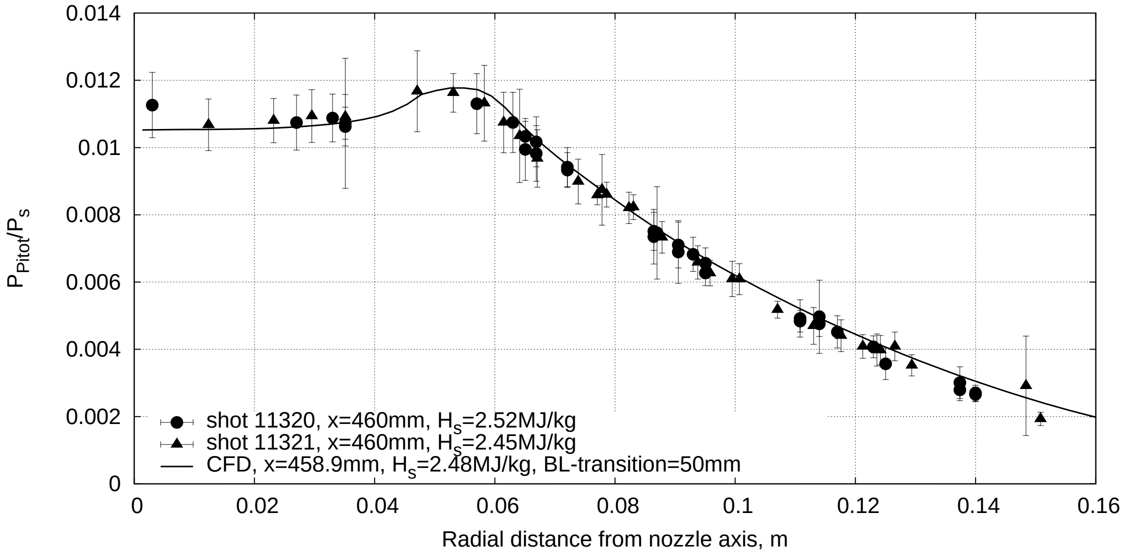


Figure 7: Pitot pressure distribution at $x = 460\,\text{mm}$.

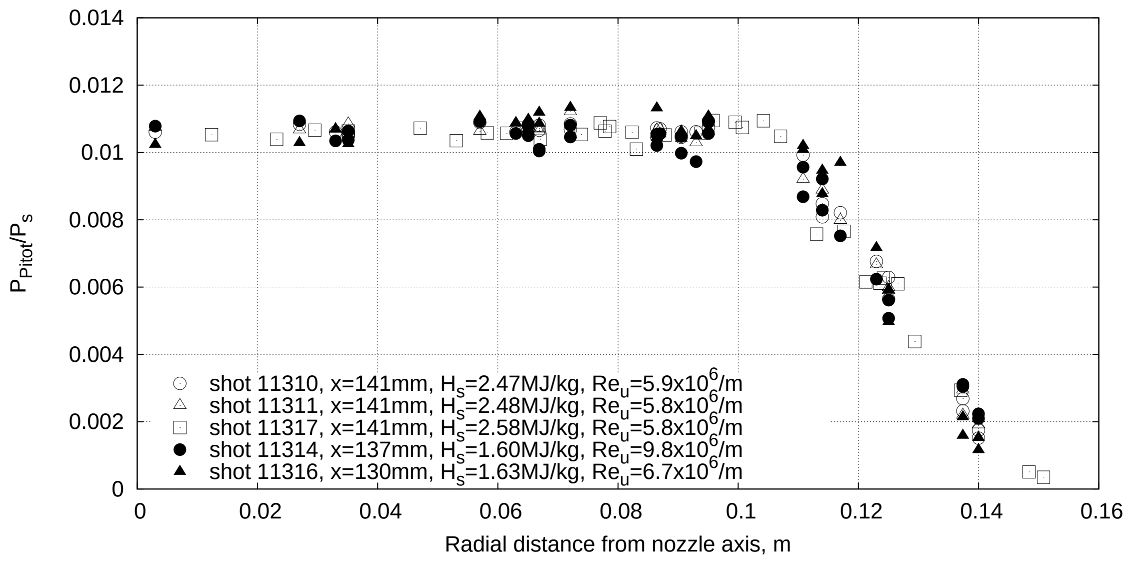


Figure 8: Pitot pressure distribution at $x \approx 140\,\text{mm}$.

for the tests at the nominal test condition - this difference is brought about by differences in the facility driver conditions for the three test conditions.

The results show that despite a 35% difference in nozzle-supply enthalpy and a 68% difference in unit Reynolds number, there are no discernable differences in the Pitot pressure distributions at $x \approx 140\,\text{mm}$. This is a good indication that the Pitot-to-nozzle-supply pressure ratio is not sensitive to changes in the test conditions (at least for the conditions tested in the current study). These results also indicate that the uniform region of the nozzle flowfield does not change in size when the test condition changes.

5 Conclusion

An excellent agreement between the experimental and numerical Pitot pressure distributions is achieved at all three locations downstream of the nozzle exit, hence demonstrating the validity of the methodology that was used to design the nozzle flowpath. The experimentally-measured Pitot pressure distribution showed that the nozzle outflow is reasonably axisymmetric and that there is good level of uniformity in the core flow region of the nozzle flowfield. Additional tests indicated that the distribution of Pitot-to-nozzle-supply pressure ratio and the size of the uniform region of the nozzle flowfield are not sensitive to changes in the nozzle-supply conditions.

References

Chan, W. Y. K., Smart, M. K., & Jacobs, P. A. (2013). Flowpath design of an axisymmetric Mach 7.0 nozzle for T4. Department of Mechanical Engineering Report 2013/02 (also UQ HIFiRE8 Technical Report Number 2013/01), The University of Queensland.

Doherty, L., Chan, W. Y. K., Jacobs, P. A., Zander, F., Gollan, R. J., & Kirchhartz, R. M. (2012). NENZFr: Non-Equilibrium NozZle Flow, Reloaded. A user guide. Department of Mechanical Engineering Report 2012/08, The University of Queensland.

Gollan, R. G. & Jacobs, P. A. (2013). About the formulation, verification and validation of the hypersonic flow solver eilmer. *International Journal for Numerical Methods in Fluids*, *73*(1), 19–57.

McGilvray, M., Jacobs, P. A., Morgan, R. G., Gollan, R. J., & Jacobs, C. M. (2009). Helmholtz resonance of Pitot pressure measurements in impulsive hypersonic test facilities. *AIAA Journal*, *47*, 2430–2439.

Smith, A. L. (1999). *Multiple component force measurement in short duration test flows.* PhD thesis, The University of Queensland, Queensland, Australia.

A Individual shot conditions

Table 1 details the facility conditions and the nozzle-supply conditions for all shots shown in this report. The notation used in Table 1 is as follows.

Res	Reservoir fill pressure
CT	Compression tube fill pressure
Ar	Volume fraction of argon in driver gas
He	Volume fraction of helium in driver gas
ST_P	Shock tube fill pressure
ST_T	Shock tube fill temperature
ST_{gas}	Shock tube fill gas type
Dia	Thickness of primary diaphragm
ss	Shock speed
p_s	Nozzle-supply pressure
T_s	Nozzle-supply temperature
H_s	Nozzle-supply enthalpy
Re_u	Unit Reynolds number

Table 1: Facility and nozzle-supply conditions for all shots shown in this report.

Shot	Res	CT	Ar	He	ST_P	ST_T	ST_{gas}	Dia	ss	recoil	p_s	T_s	H_s	Re_u
-	MPa	kPa	%	%	kPa	K	-	mm	m/s	mm	MPa	K	J/kg.K	10^6/m
11310	2.60	40.2	100	0	200	300	Air	3	1669	-119.6	19.53	2362	2.47	5.9
11311	2.60	40.2	100	0	200	300	Air	3	1679	-119.5	19.33	2371	2.48	5.8
11314	2.30	81.8	100*	0	270	300	Air	2	1305	-115.5	18.46	1707	1.60	9.8
11316	1.65	27.4	100	0	270	300	Air	2	1438	-109.0	13.02	1735	1.63	6.7
11317	2.60	40.2	100	0	200	300	Air	3	1710	-120.2	20.29	2443	2.58	5.8
11318	2.60	40.2	100	0	200	300	Air	3	1717	-120.1	20.26	2451	2.59	5.7
11319	2.60	40.2	100	0	200	300	Air	3	1694	-119.8	19.87	2408	2.53	5.8
11320	2.60	40.2	100	0	200	300	Air	3	1691	-119.3	19.79	2401	2.52	5.8
11321	2.60	40.2	100	0	200	300	Air	3	1666	-119.3	19.29	2352	2.45	5.9

* For shot 11314, the compression tube was filled with 100% nitrogen.

B Setup script for Eilmer simulation of test section

The Eilmer setup script that was used for the simulation of test section is shown in this section. Note that this script was adapted from the `nozzle.py` script that was generated from a previous NENZFr run.

```
# test-section.py
#
# Axisymmetric simulation of the T4 test section. Nozzle-exit flow
# profiles from a previous NENZFr simulation are used as inflow
# conditions to this simulation. This script is an adaptation from
# the nozzle.py script (from the NENZFr simulation).
#
# Wilson Chan, 2014
#
#--------------------------------------------------------------------------
import string, fileinput
from estcj import reflected_shock_tube_calculation as rstc
from cfpylib.gasdyn.cea2_gas import Gas, make_gas_from_name
from operator import indexOf
from cfpylib.nm.zero_solvers import secant
from string import upper, lower
from numpy import arange
from cfpylib.nm.roberts import roberts
from math import tan, atan
import os

#--------------------------------------------------------------------------
gdata.title = "Flow through a shock tunnel nozzle."

# Shock tube conditions define flow condition at the nozzle throat.
# This throat condition is used as the input to the simulation of
# the nozzle expansion.
estc_result = rstc("air", 200000.0, 300.0, 1678.62, 19325300.0, None, \
                   area_ratio=129.72, task='stn')
throat = estc_result['state6']
throat_V = estc_result['V6']
print "Flow condition at nozzle throat:"
throat.write_state(sys.stdout)

# Estimate turbulence quantities for free stream
# by specifying the intensity as 0.05 and estimating the
# turbulence viscosity as 100 times the laminar viscosity.
throat_tke = 1.5 * (throat_V * 0.05)**2
throat_mu_t = 100.0 * throat.mu
throat_omega = throat.rho * throat_tke / throat_mu_t
print "Inflow turbulence: tke=", throat_tke, "omega=", throat_omega

gastype = "air"

# Define the temperature and pressure of the initial gas sitting in
# the nozzle/dump-tank
initial = {};
initial['p'] = 106.4;
initial['T'] = 300.0;

if gastype in ['air', 'Air']:
    # Test gas is Air (CEA species) in equilibrium
    if "eq" in ['eq']:
        print 60*'-'
        print "Nozzle expansion uses Equilibrium Air (CEA LUT)"
        print 60*'-'

        select_gas_model(fname="cea-lut-air.lua.gz")
        throat_massf = [1.,]
        initial_massf = [1.,]

else:
    print "Unknown gas type specified"
```

```
# Now set the inflow and intial flow conditions
inflow=FlowCondition(p=throat.p, u=throat_V, v=0.0, T=throat.T,
                     massf=throat_massf, tke=throat_tke,
                     omega=throat_omega)
initial=FlowCondition(p=initial['p'], u=0.0, v=0.0, T=initial['T'],
                      massf=initial_massf, tke=0.0, omega=1.0,
                      label="initial")


# When no gridFileName is given we use the provided (or default) contourFile
if "None" in ['None']:
    # Read in contour file for the particular nozzle.
    x_c = []; y_c = []
    fp = open("Bezier-control-pts-t4-m7.data", 'r')
    for line in fp.readlines():
        items = line.strip().split()
        if len(items) == 0: continue  # skip blank lines
        if items[0] == '#': continue  # skip header lines
        x_c.append(float(items[0]))
        y_c.append(float(items[1]))
    area_ratio = (y_c[-1]/y_c[0])**2
    print "nozzle area ratio=", area_ratio

    # Bezier control points that make up the nozzle contour.
    if "Bezier-control-pts-t4-m7.data" in ['Bezier-control-pts-t4-m7.data']:
        # Only for the T4 Mach 8 nozzle, the control points start
        # from the first pair of coordinates in the given data file.
        bezCtrlPts = [Vector(x_c[i],y_c[i]) for i in range(len(x_c))]
    else:
        # The first pair of coordinates in the given data file for
        # the other nozzles are not Bezier control points.
        bezCtrlPts = [Vector(x_c[i],y_c[i]) for i in range(1, len(x_c))]

    # Create nodes
    throat_axis = Node(-y_c[0], 0.0)
    throat_wall = Node(-y_c[0], y_c[0])
    divergence_axis = Node(0.0, 0.0)
    divergence_wall = Node(0.0, y_c[0])
    nozzle_end_axis = Node(x_c[-1], 0.0)
    nozzle_end_wall = Node(x_c[-1], y_c[-1])

    # Define the lines making up the patch for the throat ..
    t_south = Line(throat_axis, divergence_axis)
    t_north = Line(throat_wall, divergence_wall)
    t_west = Line(throat_axis, throat_wall)
    t_east = Line(divergence_axis, divergence_wall)
    # .. and the expansion region.
    if "Bezier-control-pts-t4-m7.data" in ['Bezier-control-pts-t4-m7.data']:
        n_north = Bezier(bezCtrlPts)
    else:
        n_north = Polyline([Line(divergence_wall,bezCtrlPts[0]), Bezier(bezCtrlPts)])
    n_west = Line(divergence_axis, divergence_wall)
    n_east = Line(nozzle_end_axis, nozzle_end_wall)

    # Define the expansion_region using a specialised Surface Function.
    # For viscous simulations, it is necessary to keep the cells near
    # the non-slip walls as orthogonal to the walls as possible. However,
    # because the "AO" option in make_patch() does not give a grid that is
    # good enough for the nozzle geometry, a specialised surface function
    # has to be used. Points in the grid along the north, east and west
    # edges follow that specified by n_north, n_east and n_west. The rest
    # of the other points in the grid are built by creating strips of
    # quadratic Bezier curves that run from the nozzle wall to the axis.
    # The use of quadratic Bezier curves allows the generated points to
    # be orthogonal to the wall near the nozzle wall and orthogonal to
    # the axis near the axis.
    def make_expansion_region_grid(r, s):
        global n_north, n_west, n_east
        if r == 0.0:
            x = n_north.eval(r).x
            y = n_west.eval(s).y
        elif r == 1.0:
            x = n_north.eval(r).x
            y = n_east.eval(s).y
        elif s == 1.0:
            x = n_north.eval(r).x
            y = n_north.eval(r).y
        else:
            # Wall point (Bezier control point 1)
            wall_pt_x = n_north.eval(r).x
            wall_pt_y = n_north.eval(r).y
            # Angle perpendicular to the wall at wall point
            wall_angle = atan((n_north.eval(r+0.0001).y - n_north.eval(r-0.0001).y) /\
                              (n_north.eval(r+0.0001).x - n_north.eval(r-0.0001).x))
            # If the expansion region starts sharply from the throat, then we
            # need some way of transitioning from the grid in the throat region
            # to the expansion region. To do so, we tweak the wall_angle in a
            # small starting region in the nozzle (say, 2% of the length of the
            # expansion region). The wall_angle starts from 0 degrees at the
            # start of this small region and smooths it out to the actual
            # wall_angle at the end of this small region.
            north_length = n_north.eval(1.0).x - n_north.eval(0.0).x
            if (n_north.eval(r).x - n_north.eval(0.0).x) <= (0.02 * north_length):
                wall_angle = (n_north.eval(r).x - n_north.eval(0.0).x) / \
                             (0.02 * north_length) * wall_angle
            # Do the same for a small region at the end of the nozzle. This is
            # to accommodate to nozzles that have been a non-zero gradient at
            # at the nozzle exit (which is brought about by nozzle truncation).
            if (n_north.eval(r).x - n_north.eval(0.0).x) >= (0.98 * north_length):
                wall_angle = (n_north.eval(1.0).x - n_north.eval(r).x) / \
                             (0.02 * north_length) * wall_angle
            # Mid point (Bezier control point 2).
            mid_pt_y = n_north.eval(r).y / 2.0
            mid_pt_x = wall_pt_x + (mid_pt_y * tan(wall_angle))
            # Axis point (Bezier control point 3).
            axis_pt_x = mid_pt_x
            axis_pt_y = 0.0
            # Generate t for quadratic Bezier curve equation.
            t = (1.0 - s)
            # Generate point on quadratic Bezier curve.
            x = (1-t)*(1-t)*wall_pt_x + 2*t*(1-t)*mid_pt_x + t*t*axis_pt_x
            y = (1-t)*(1-t)*wall_pt_y + 2*t*(1-t)*mid_pt_y + t*t*axis_pt_y
        return (x, y, 0.0)
    #
    expansion_region = PyFunctionSurface(make_expansion_region_grid)

    # Define the clustering and grid resolution parameters for the "nozzle_blk".
    # We define these here like this as they may be needed to define the
    # "throat_blk" for the Mach 4, 6 and 8 nozzles
    nnj = 80   # number of radial cells
    nni = 600   # number of axial cells
    nbi = 30   # number of axial blocks for the divergence section (nozzle_blk)
    nbj = 4   # number of radial blocks
    bx = 1.5     # clustering in the axial direction
    by = 1.0012     # clustering in the radial direction
    x_clust = RobertsClusterFunction(1, 1, bx)
    y_clust = RobertsClusterFunction(0, 1, by)
    cf_list = [x_clust, y_clust, x_clust, y_clust]

    # Define boundary conditions
    Twall = 300.0
    bc_list = [FixedTBC(Twall), ExtrapolateOutBC(), SlipWallBC(), SupInBC(inflow)]

    if "Bezier-control-pts-t4-m7.data" not in ['Bezier-control-pts-t4-m10.data']:
        # Now, we also make a block for the throat region. We must define this
        # block here BEFORE the "nozzle_blk" in order to ensure that the blocks are
        # numbered in sequence in order to allow space-marching in the simulation.
        #
        # The number of cells in the axial direction and clustering for
        # this block are automatically calculated based on the clustering and
        # resolution of "nozzle_blk" as specified above. We try to ensure that the
        # spacing across the interface of the "throat_blk" and "nozzle_blk" is even
        # and an appropriate number of cells/blocks are used in the throat.
        #     Luke D. 21-Aug-2011
        throat_region = make_patch(t_north, t_east, t_south, t_west)

        # We need to know the axial spacing of the first grid point in
        # "nozzle_blk" in order to calculate the required clustering.
        # The following code isn't very elegant but it works.
        delta = 1.0/float(nni) # Intial uniform spacing
        # Non-uniform spacing in logical space for the SOUTH boundary of "nozzle_blk"
        logical_noz = roberts(arange(0.0, 1.+delta, delta), 0.5, bx)
        # Acutal grid points
        actual_noz = x_c[-1] * logical_noz
        diff = actual_noz[1] - actual_noz[0]

        # Calculate the number of cells required on the NORTH/SOUTH boundaries of the
        # "throat_blk" based on how many cells there are in the "nozzle_blk" over the
        # same physical length. NB. If you are clustering very strongly in the x-direction
        # (x_clust, bx specified above) you may want to include a factor here to reduce
        # the number of cells in the throat region.
        nni_throat = [indexOf(actual_noz, x) for x in actual_noz if x < y_c[0]][-1] + 1
        delta2 = 1./nni_throat # Intial uniform spacing
        logical_throat = arange(0.0, 1.+delta2, delta2)

        # Define a function which calculates the difference in the grid
        # spacing on either side of the "throat_blk"/"nozzle_blk" boundary
        def spacing_error(x, L=y_c[0], diff=diff, orig=logical_throat):
            new = L * roberts(orig, 0.5, x)
            return (new[1]-new[0] - diff)/diff
        bx_throat = secant(spacing_error, 1.0001, 1.002, tol=1.0e-2,\
                           limits=[1.0001,5])
        if bx_throat == 'Fail' or 'FAIL':
            print "Failed to find throat clustering parameter"
            print "Setting bx_throat == bx"
            bx_throat = bx
        # Infer the number of blocks from how many we set for "nozzle_blk"
        nbi_throat = int(round(float(nbi)/float(nni)*float(nni_throat)))
        if nbi_throat > 1:
            nbi_throat = nbi_throat - 1
        # We need at least 1 block for the throat region
        if nbi_throat < 1:
            nbi_throat = 1

        # Display the calculated data
        print "nni_throat=",nni_throat
        print "nbi_throat=",nbi_throat
        print "Clustering, bx_throat=",bx_throat

        # Now set the clustering and define the "throat_blk"
        x_clust_throat = RobertsClusterFunction(1, 1, bx_throat)
        cf_list_throat = [x_clust_throat, y_clust, x_clust_throat, y_clust]


    #-----------------------------------------------------------------------------------------
    # Additional part in the NENZFr-generated script to simulate the test section flow.
    #
    import sys
    sys.path.append(os.path.expandvars("$HOME/e3bin")) # installation directory
    from e3_flow import *

    # Some variables needed for the test section simulation
    workingDir = os.getcwd()  # current working direction
        # directory that contains the NENZFr simulation
    previousJobDir = "../fine-grid-run-of-iteration-115-BLTrans-50mm-shot-11311-decrease-yplus/"
    jobName = "nozzle"  # job name of the NENZFr simulation
    tindx = 9999  # time index where the solution of the NENZFr simulation is
    test_section_length = 0.47
    test_section_half_height = 0.225  # equivalent half-height in this case
    ts_nbi = 6

    def locate_start_of_last_nozzle_blocks(jobName, nblock, nbj, tindx):
        # Read in all grid and flow files
        zipFiles = 1
        grid, flow, dimensions = read_all_blocks(jobName, nblock, tindx, zipFiles)
        # Find the x-location associated with the start of the last column of nozzle blocks
        nozzle_start = grid[-nbj].x[0][0][0]
        nozzle_nni = flow[-nbj].ni
        return nozzle_start, nozzle_nni

    # Change in the the NENZFr simulation directory, locate the start of
    # the last column of blocks in the already-computed NENZFr simulation,
    # and return back to the current working directory.
    os.chdir(previousJobDir)
    nblock = nbi*nbj + nbi_throat*nbj
    nozzle_start, nozzle_nni = locate_start_of_last_nozzle_blocks(jobName, nblock, nbj, tindx)
    os.chdir(workingDir)

    # Use root-finder to locate the r0 of the expansion region that
    # corresponds to the x-location of the start of the last column
    # of the already-computed nozzle simulation.
    def error_in_r(r):
        global nozzle_start
        return (nozzle_start - expansion_region.eval(r, 0.0).x)
    r0 = secant(error_in_r, 0.9, 0.91, tol=1e-12, limits=[0.0,1.0])

    # With r0 found, we can now reset the extent of the expansion_region
    # grid, such that we are only re-computing a small portion of the end
    # of the nozzle.
    expansion_region.r0 = r0

    # Create the nozzle block
    nozzle_blk = SuperBlock2D(expansion_region,
                              nni=nozzle_nni, nnj=nnj, nbi=1, nbj=nbj,
                              bc_list=[FixedTBC(Twall), AdjacentBC(),
                                       SlipWallBC(), StaticProfBC(n_profile=2)],
                              cf_list = cf_list,
                              fill_condition=initial, label="nozzle")

    # Create node points for the test-section
    test_section_start_axis = nozzle_end_axis
    test_section_start_intermediate = nozzle_end_wall
    test_section_start_wall = Node(test_section_start_axis.x, test_section_half_height)
    test_section_end_axis = Node(test_section_length + test_section_start_axis.x,
                                 test_section_start_axis.y)
    test_section_end_intermediate = Node(test_section_length + test_section_start_axis.x,
                                         test_section_start_intermediate.y)
    test_section_end_wall =  Node(test_section_length + test_section_start_axis.x,
                                  test_section_start_wall.y)

    # Build lines from the created nodes
    ts_core_north = Line(test_section_start_intermediate, test_section_end_intermediate)
    ts_core_east = Line(test_section_end_axis, test_section_end_intermediate)
    ts_core_south = Line(test_section_start_axis, test_section_end_axis)
    ts_core_west = Line(test_section_start_axis, test_section_start_intermediate)
    ts_wall_north = Line(test_section_start_wall, test_section_end_wall)
    ts_wall_east = Line(test_section_end_intermediate, test_section_end_wall)
    ts_wall_south = ts_core_north
    ts_wall_west = Line(test_section_start_intermediate, test_section_start_wall)

    # Build surfaces from the created lines
    ts_core_region = make_patch(ts_core_north, ts_core_east, ts_core_south, ts_core_west)
    ts_wall_region = make_patch(ts_wall_north, ts_wall_east, ts_wall_south, ts_wall_west)

    # Build blocks from the created surfaces
    x_clust_ts = RobertsClusterFunction(1, 0, 1.5)
    ts_nni = int(round(test_section_length/(nozzle_end_axis.x-nozzle_start)*nozzle_nni))
    ts_core_blk = SuperBlock2D(ts_core_region, nni=ts_nni, nnj=nnj, nbi=ts_nbi, nbj=nbj,
                               bc_list=[AdjacentBC(), ExtrapolateOutBC(),
                                        SlipWallBC(), AdjacentBC()],
                               cf_list = [x_clust_ts, RobertsClusterFunction(0, 1, 1.15),
                                          x_clust_ts, y_clust],
                               fill_condition=initial, label="test-section-core")
    #
    ts_wall_nnj = int(round((test_section_half_height-nozzle_end_wall.y)/nozzle_end_wall.y*nnj))
    ts_wall_blk = SuperBlock2D(ts_wall_region, nni=ts_nni, nnj=ts_wall_nnj, nbi=ts_nbi, nbj=2,
                               bc_list=[SlipWallBC(), ExtrapolateOutBC(),
                                        AdjacentBC(), SlipWallBC()],
                               cf_list = [x_clust_ts, RobertsClusterFunction(1, 1, 1.09),
                                          x_clust_ts, RobertsClusterFunction(1, 0, by)],
                                      fill_condition=initial, label="test-section-wall")
        #
        identify_block_connections()


    # Attributes for the space-marching simulation.
    gdata.dimensions = 2
    gdata.sequence_blocks = 0
    gdata.axisymmetric_flag = 1
    gdata.viscous_flag = 0
    gdata.flux_calc = ADAPTIVE
    gdata.max_time = 0.005  # seconds
    gdata.max_step = 1000000
    gdata.dt = 0.2e-9
    gdata.cfl = 0.4
    gdata.cfl_count = 5
    gdata.dt_plot = 0.04e-3
    gdata.dt_history = 10.0e-6
    gdata.turbulence_flag = 1
    gdata.turbulence_model = 'k_omega'

    gdata.max_time = 0.002  # seconds
    gdata.max_step = 999999999
    gdata.dt_plot = 0.5e-3
```