University of Sheffield

Aerospace Engineering

Final Year Project

# Design of a Modular, Multifuel Test Stand for Liquid Bipropellant Rockets


Emma C. L. Patterson

*Supervisor:* Alistair John

A report submitted in partial fulfilment of the requirements
for the degree of Master of Engineering

# Chapter I

# Acknowledgements

Completing this dissertation marks one of the final parts of my studies at the University of Sheffield and a period of my life that has shaped me immensely. I am thankful to my supervisor, Dr Alistair John, for his support throughout the writing of this report and for his work within rocketry and student liquid propulsion in the UK. My thanks also go to Sunride, Race2Space, and university rocketry teams across the UK for the environment of friendly competition and support they have fostered. The growth I have seen in student rocketry over the last 5 years is incredibly exciting for the future of engineering and space across Europe. I am deeply thankful to my family for their unwavering support during my degree and the writing of this thesis, and for always encouraging me to aim higher. I hope the work that has gone into this thesis can contribute to future testing at Buxton, and that the tools presented herein can be a final contribution to the progress of the Sunride Team.

# II Table of Contents

# Chapter III

# Nomenclature and Abbreviations

| | |
|---|---|
| $F$ | Force |
| F9 | Falcon 9 (vehicle) |
| FD | Flame Deflector |
| FEA | Finite Element Analysis |
| $h$ | Heat Transfer Coefficient |
| IPA | Isopropyl Alcohol |
| ISP | Specific Impulse |
| LC | Load Cell |
| LNG | Liquid Natural Gas |
| LOX | Liquid Oxygen |
| LPG | Liquid Propane Gas |
| NASA | National Aeronautics and Space Administration |
| OF ratio | Oxidiser to Fuel Ratio |
| P&ID | Piping and Instrumentation Diagram |
| $q$ | Heat Flux |
| RCS | Reaction Control System |
| RPA | Rocket Propulsion Analysis |
| SLS | Space Launch System |
| SpaceX | Space Exploration Technologies |
| SPL | Sunride Propulsion Lab |
| SRB | Solid Rocket Booster |
| STP | Standard Temperature and Pressure |
| $T$ | Thrust |
| TF | Thrust Frame |
| TVC | Thrust Vector Control |
| $\delta$ | Impingement Angle |
| $\Theta$ | Uplift Angle |
| $\dot{m}$ | Mass flow Rate |

# IV List of Figures

# V List of Tables

# Chapter VI

# Abstract

Student-developed rockets allow students to explore real engineering lifecycles, manage teams and budgets, and build hands-on experience. In the UK, liquid rocketry has had a recent surge in popularity due to the Race2Space competition making testing more accessible, leading to the development of several liquid-powered rockets by Sunride and other British student groups. To allow the scope and scale of these rockets to keep increasing in the coming years, testing of the feed systems through vertical hot fires will be essential. This project produces piping and instrumentation diagrams (P&ID) for several propellants, designs for the structural elements of the test stand, and designs and produces a subscale flame deflector. Additionally, the FLAME program has been developed to analyse the heating from engine exhaust plumes, which allows rapid assessment of new engines for testing. .

# Chapter 1

# Introduction

For the last approximately 10 years there has been an expansion in student and amateur groups worldwide building liquid-fuelled rocket engines and vehicles. Recently, Sunride became the first British team to launch a liquid rocket.
There are three design challenges to liquid rocket production: design of the avionics, recovery, and aerodynamics; design of the engine; and design of the tank and feed system. The first is a challenge shared with solid rockets, and there is a pool of talent to draw from. Engine design is also increasingly common, with Race2Space leading to dozens of UK University teams with experience in the design and hot-firing of bipropellant liquid rocket engines. The tank and feed design is less established, and testing is essential to allow rapid prototyping and a "fail-fast" methodology, allowing a high rate of progression.
Currently, there are just 2 British student teams with access to a dedicated test site, and this project aims to make the Sunride the first team with support for multiple fuels, including cryogens. A student-led alternative to costly private test sites or once-a-year, engine-only testing would improve access to these facilities for university students in the surrounding area, and this is hoped to have a positive impact on student rocketry in the UK.
Student rocketry projects are impacted by the fact that the student body is in constant flux. Academic supervisors are the only members to be involved for more than a handful of years, leading to challenges with knowledge transfer and training new members. These new members may have different views on the development path, making the projects being undertaken in 2-5 years time nebulous at best. It is for this key reason that a modular test stand is being investigated. There is a promising design direction in the current IPA-Nitrous test setup, but alternate propellant mixtures such as IPA-LOX, Ethane-Nitrous, Propane/Methane-LOX, and others have enough promise that vehicles using these propellants are foreseeable. It is therefore advantageous to the Sunride team to ensure there is a flexible platform to support the widest range of propellant combinations and vehicle specifications possible.
There are several types of testing that could utilise a modular test stand. Vehicle hot-fire tests are likely to continue using Nitrous Oxide. Engine and Turbo-machinery tests may instead utilise Oxygen in either a liquid or gaseous form. Either of these oxidisers could be used in conjunction with several different fuels, so a fully modular test stand can effectively support future engines regardless of changes to design direction.

## 1.1 Background

The testing site available to the University of Sheffield is located on the Blastech Ltd. site near Buxton, in the Peak District. Throughout 2024 and 25, procurement and assembly work has been underway on the oxidiser and fuel delivery, mechanical interface, and Data Acquisition (DAQ) systems. The designs for a horizontal test stand detailed by Bentley[3] have been manufactured and implemented, and the first hot fire tests of Sunride engines are planned to take

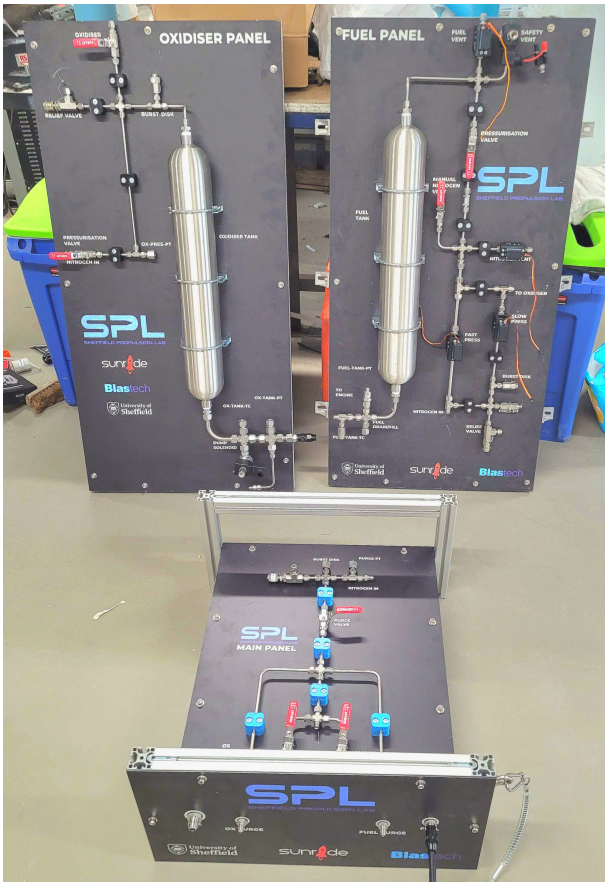



Figure 1.1: Oxidiser and Fuel panels for SPL Test Rig

place in early-mid 2025.
The Sunride team has continued the development of both liquid bipropellent engines and rockets, with test flights of Black Adder, the first student liquid rocket to launch within the UK, having taken place throughout the first months of 2025. Black Adder is powered by a scaled-down $1.25kN$ regeneratively cooled aluminium engine, Sunfire IV, designed from the ground up for flight operations. The capacity for vertical hot fire testing will simplify the testing and development of both Black Adder and future liquid rockets.

# Chapter 2

# Aims and Objectives

This project aims to provide designs for all the hardware needed to upgrade the test site, allowing multiple fuel combinations, including cryogens. This will allow the Sunride Rocketry Team to conduct in-house testing of engines and rockets with a variety of propellants, with sufficient capacity for the foreseeable future of development within the team. The first section of this work will be to define what those future requirements will be. This includes the propellant selection and flow rate calculations, which will determine the upper limit of the force on the structure of the test stand. The next step is the detailed design of the test stand, key hardware, and producing the piping and instrumentation diagrams (P&ID) of each propellent. Finally, preparations for test verification of a scale model flame deflector (FD) system through the development of heat flux prediction techniques.
Key deliverables will include:

- Thrust Frame Design
- Flame Deflector Concept Design & Manufacturing of Scale Test Model
- Flame Deflector Environment Modelling software (FLAME) & Thermal FEA Model
- LOX Oxidiser Module P&ID Design
- Kerosene, IPA, Methane/Propane Fuel P&ID Module Designs
- Pneumatic Module P&ID Design

Design of the FD will include the creation of a model to predict the thermal environment encountered due to the engine plume. The composition, speed, temperature etc. of the plume will vary depending on propellent, thrust level, chamber pressure, OF ratio, and other engine parameters. Because of this, a computationally light program to estimate the heat flux for an arbitrary engine is desirable, reducing the risk of damage to the FD and ensuring safe operation. The TF will be designed and analysed to ensure it is fit for purpose, and the high-level design of all propellent modules will be complete

# Chapter 3

# Literature Review

## 3.1 Test site design

Commercial test sites will usually have one propellant combination, depending on the vehicle being tested. There are companies focusing on creating fully modular test sites for use with multiple vehicles, namely SpaceDreams' NuPad, which is still in the concept stages. Other commercial and research test sites are therefore investigated to inform the design of the mechanical and fluidic systems here. SpaceX test sites are varied in their construction, with some sites purpose-built and others retrofitted by SpaceX. Both stage and engine testing facilities will be analysed, including first and second stage Falcon 9 (F9) test facilities, Merlin engine vertical test stands, and Raptor engine vertical and horizontal test stands.
First stage F9 testing can be seen to use cable supports. This is a cost-effective method of securing the stage using the tension in the lines. It allows variation from vehicle to vehicle in length of the first stage tanks. Different stages are different heights, due to improvements that has been made throughout the development of the F9. It is a unique factor in having to re-test older stages after recovery, and cable support could be explored for the retention of various size tanks during stage hot-fires. Testing of both engine types happens both horizontally and vertically. Multiple vertical testing sites are converted from use for stage tests, as can be seen on the 'Tripod Stand', however some purpose-built sites also feature vertical engine-only test stands. There are advantages to testing in the flight orientation: TVC can be implemented more easily; there is no risk of pooling of propellents in coolant or injector volumes; along with other advantages. Horizontal test stands can be seen with links that could be load cells (LCs); the cells may be integrated into the structure in a location not visible from the available camera angles.

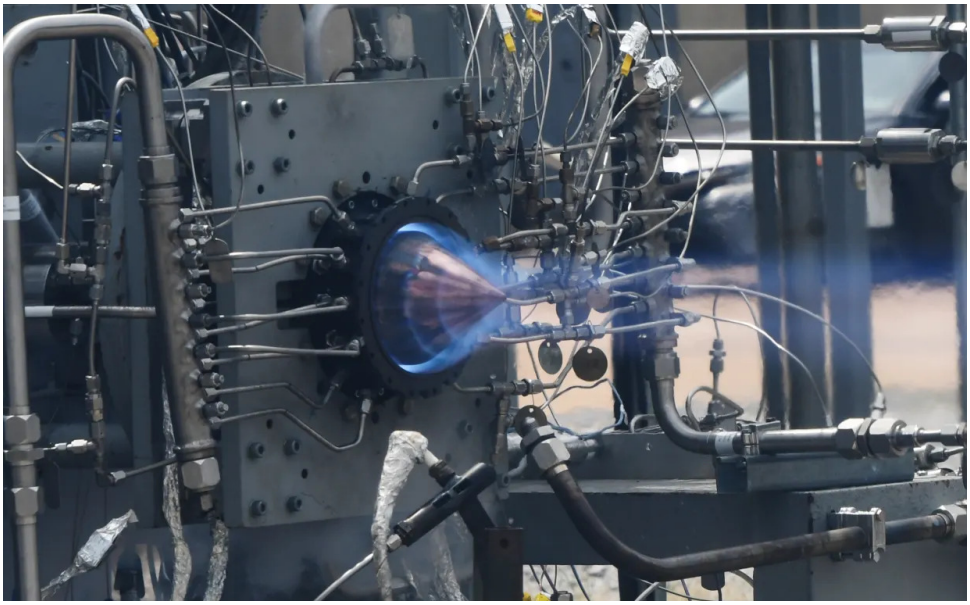


Figure 3.1: NASA hot fire testing of an RDE at the Marshall Space Flight Centre

Testing of the RS-25 engine by NASA uses a thrust adapter to meet the engine interfaces. The LCs are designed to function as the interface with this thrust adapter and the engine. This requires significant design investment and cost [4] and as such is impractical at this scale. Other NASA tests can be seen in 3.1 using a plate with mounts for the engine and 4 load cells, likely button type. This places the LCs in shear from the weight of the test article. RS-25 testing is done vertically, with the Rotating Detonation Engine (RDE) testing being horizontal. This is likely due to the RDE being designed just for testing, meaning that the horizontal firing direction can be accounted for in the design. Flight engines are, however, tested vertically in line with their operational mode.

## 3.2 Flame Deflector Design

A key component of vertical testing is the flame deflector used to control the plume of the engine. Supersonic flows of high temperature gas leads to a large amount of damage and erosion to the ground and standard concrete surfaces when the plume impinges upon it. Some fuels lead to the plum being corrosive, posing an additional design challenge. Impingement of the plume on surfaces creates intensely damaging acoustic energy and can lead to the ejection of debris that can damage the test article.
There are multiple design approaches used for flame deflectors. Active cooling allows more convenient materials to be used beyond their standard limits. Ablatives are designed to be replaced between uses. Refractory concretes can survive the environment thermal environment without damage. Any of these methods can also be combined with plume water injection to improve their effectiveness.
Flame trenches are a somewhat common addition to dedicated orbital launch sites, usually used by government-backed test and launch sites. Their is a reluctance to use them by private installations due to an incredibly high cost of the earthworks needed to install them. The usual method of construction involves building up a mound of soil atop which the launch pad is placed, with the trench being lined with a refractory concrete. Occasionally a flame deflector using another technology can be installed within the trench, as was seen for the main engines and solid boosters used for the Space Shuttle. As testing sites have become more common, above ground deflectors have been used, shown below in 3.2. The flame deflectors used by SpaceX for testing have a unique appearance, as they are constructed from tubes. These tubes

Figure 3.2: NASA test of RS-25 engine at Stennis Space Centre with the B2 test stand, utilising a water cooled flame deflector with plume injection

are welded together, and have water flowing through them. The water enters at the exit end of the deflector and travels upwards. At the top of the deflector (nearer to the engine) the pipes seem to stop, several uses for the water are proposed. Film cooling could be used, small nozzles placed across the surface of the deflector would produce and the layer of steam, protecting the surface of the deflector. Alternatively, larger nozzles in key areas could act similar to plume injection techniques. Finally, the pipe sections could be connected with an outlet that is not readily visible. The cross-section of the deflectors used are bucket-shaped.

The deflector used for the larger starship and superheavy booster uses similar principles, with a flat plate positioned at 90° to the vertical axis of the vehicle with water jets rising up, meeting the plume and protecting the surface.

Historically, the favoured approach for launch sites and test sites has been some form of ablative or refractory deflector. Refractory materials maintain their properties above the temperature at which steel would melt. There are several types detailed in [5], which are evaluated for their environmental resistance to the marine environment of the launch site, the corrosive environment of the shuttle SRB exhaust, as well as the thermal environment. The main surface of the deflector is composed of a refractory concrete using primarily calcium aluminate aggregate. Additional coatings for exposed steel surfaces include thermal epoxies and ablative silicones.

NASA's SLS uses replaceable 3" thick steel plates for the surface of the deflector. This saw significant wear during the first launch of SLS, with over 1" of ablation on the high heat areas [6]. SLS static fire testing was conducted at NASA's B2 Test stand in Mississippi, which uses a water-cooled deflector, as well as plume injection. The refractory surface of the deflectors of the Saturn program saw significantly less wear[2]. It is worth noting that SLS and Space Shuttle deflectors are used for launches, and so the duration of exposure to the plume is significantly less than for test sites.

Uncooled refractory or ablative deflectors are more commonly used in launch sites than test sites, as the duration of exposure is on the order of seconds. For long duration hot fires in the case of single engine long burn tests, or second stage tests, thermal steady state may be reached, and the materials of the deflector and the cooling system used will have to be sufficient for the

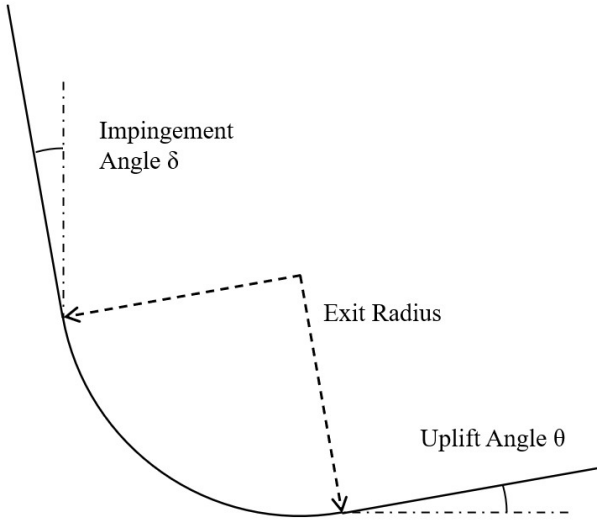



Figure 3.3: Sections of a flame deflector, redrawn from [1]

engine(s) used.
Plume injection is a very common [7, 8, 9, 10, 11, 12] way to improve the thermal protection of deflectors. The primary cooling effect is the vaporisation of the water [13] and reducing the after-burning effect within the plume [14], which can be a significant source of heat [15, 16, 17]. An additional function is the reduction in acoustic energy [18], which can be a significant source of damage to brittle materials such as refractory concrete. The primary downsides is increasing the mass of gas within the plume, increasing the forces exerted on the deflector, shown in [2] to depend on the mass flow by the following relationship:

$$F_{V_1} = 2T sin^2(\delta) \tag{3.1}$$

$$F_{H_1} = \frac{T}{2} sin(2\delta) + T cos(\delta)(1 - sin(\delta)) \tag{3.2}$$

$$F_{N_2} = T \cos\delta \sin\theta \tag{3.3}$$

$$F_{T_2} = T \cos\delta (1 - \cos\theta) \tag{3.4}$$

$$F_{H_2} = F_{N_2} \cos\delta - F_{T_2} \sin\delta \tag{3.5}$$

$$F_{V_2} = F_{T_2} \cos\delta + F_{N_2} \sin\delta \tag{3.6}$$

$$T = \frac{\dot{m} v_{ex}}{g} = \frac{(\dot{m}_{engine} + \dot{m}_{water})\, v_{Plume\, after\, injection}}{g} \tag{3.7}$$

A downside to plume injection is the increased infrastructure required to pump incredibly large amounts of water. For the NASA B2 test stand this is on the order of 15 tonnes per second, higher than the flow rate of all the turbo-pumps within the engines combined. The power requirements for this volume flow rate is significant, such that electric pumps are usually impractical; size and power infrastructure advantages making diesel pumps preferred.

### 3.2.1 Flame Deflector Thermal Modelling

There is a large body of research around subsonic jet impingement problems, for example [19], and for supersonic rocket plumes [20]. The usual method for design of a flame deflector is by computational methods, as seen in [21, 22, 23]. There are a few analytical approaches to rocket plume heat flux, the first group is applicable to spacecraft applications. The heating caused by plume impingement on surfaces of spacecraft from their own reaction control system (RCS)

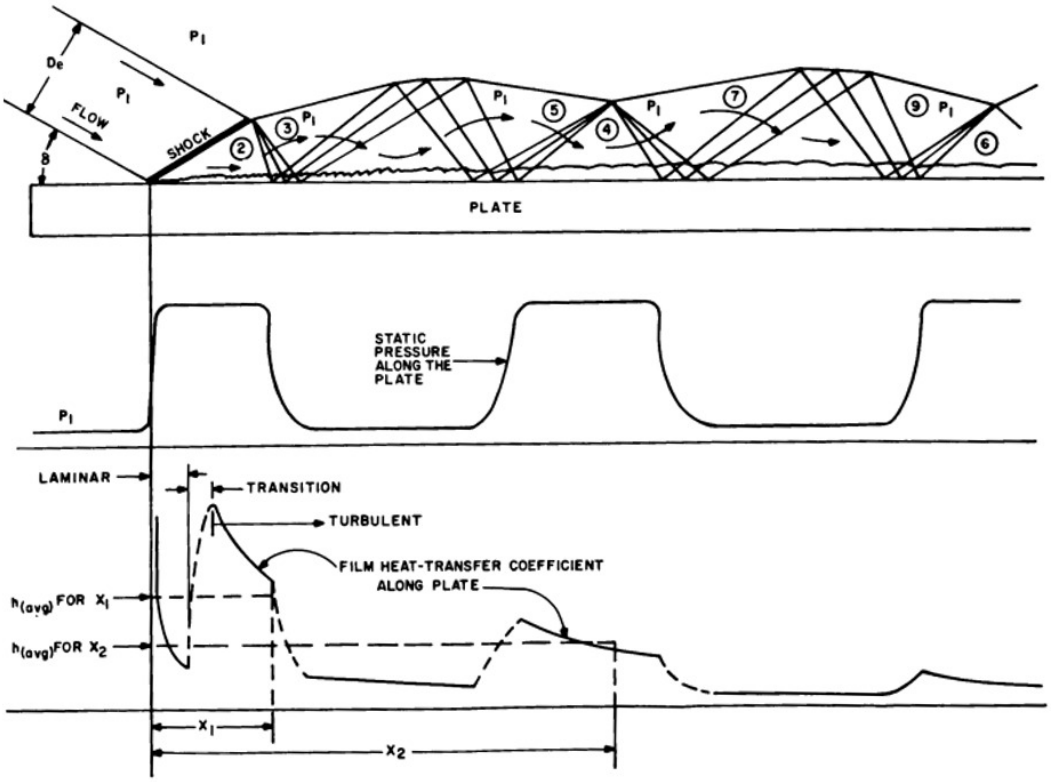



Figure 3.4: Shock structure, pressure distribution, and heat transfer rate across the impingement zone, reproduced from Evans and Sparks [2]

thrusters are evaluated by Prickett in [24], and measured experimentally in [25]. Prickett's method is shown to be accurate when the correct experimental correlation is included, however, the approach uses assumptions of a rarefied environment. This means supersonic effects are not accounted for; the maximum heating condition considered is that of flow parallel to the surface of interest, as direct impingement is not considered. Because of the lack of impingement, the heat flux is likely to be underestimated when compared to other techniques.

Evans and Sparks [2] present an analytical approach to the problem at hand, evaluating the conditions after an oblique shock forms at the impingement point and using correlations for flow along a flat plate to evaluate the heat flux across a given section of the FD. This approach is a simple to implement, the oblique shock angle can be found numerically and the flow conditions evaluated using shock relationships. This technique finds the forced convective heat transfer coefficient, $h$. Evans and Sparks do not, however, account for the maximum heating rate along the FD, which would be at the point of attachment of the oblique shock. This can be seen in 3.4 with the initial heat transfer rate being shown as an asymptote.
A NASA standard [1] includes a relationship to find the maximum temperature of the surface of the FD. This relationship is dependent on the heat transfer rate of the exhaust plume, thermal properties of the FD, and the duration of the exposure of the FD. The total heat transfer rate is based on the convective, radiant, and particle contributions. The convective rate, $q_c$ is acquired experimentally, with the radiant rate, $q_r$ fixed at 20% of $q_{total}$. The particle heat transfer rate, $q_p$ can be assumed to be relevant only for solid rocket motors, as liquid propellants included do not produce metallic particulates in their exhaust plume. Alternate methods for estimating the heating caused by flow of the plume over a surface include those used to evaluate nozzle heating. Bartz [26] describes a method for quickly evaluating the nozzle heating, and is widely used in rocket nozzle design and development. This relationship is used in [27] and is implemented in

the Rocket Propulsion analysis (RPA) program.
None of these solutions find the heat flux at the point of attachment of the shock. Eckert [28] does present a solution. This solution finds the coefficient of friction of the flow, uses a Prandtl number approximation to find the recovery factor, and then a Stanton number approximation to find the heat transfer rate. This provides a value for the heat flux from the attachment point of the shock. The test data that these relationships were produced used cold gases, so there may be issues when applied to a plume environment.

## 3.3 Current Test Infrastructure

Work completed by Bentley [3] has formed the basis of current testing infrastructure, this includes P&ID design of a nitrous oxide and IPA test rig, structural design, data acquisition (DAQ) specification, and control system. Lots of this work does not need to be repeated, that being the DAQ and control systems, as the existing designs can be expanded to allow the increased number of sensors required for cryogenic operations. The structual design is solely focused on horizontal testing up to a capacity of $5kN$. The thrust frame can still be used for lower capacity horizontal testing. The nitrous oxide delivery system has a maximum mass flow rate of 2kg/s, based on the same performance criteria as here, so can be reused. The IPA system is too low capacity to satisfy the requirements of a maximum LOX flow rate engine, so shall be redesigned.

## 3.4 Literature Review Key Considerations

There are several key pieces of knowledge that have been taken from this review of the literature. With regards to test site design this includes:

- Enabling stage testing to utilise tensioned cables or guidelines to secure the stage.
- Cooling flame deflectors can be done multiple ways, including plume injection and regenerative cooling of the deflector
- Finding the heating rate of the flame deflector by CFD is incredibly complex, and an analytical solution would speed up testing
- Combining the approaches of several analytical models could increase accuracy

# Chapter 4

# Requirements Definition

The overriding requirement is that the system is possible to be manufactured by a student team. This means standard componentry and simple construction methods should be used wherever possible, that cost should remain reasonable, and documentation must be sufficient to allow the construction of modules at a later date without the author's assistance. The first constraint this requirement places is the size of propellant delivery equipment, which in turn limits the maximum propellent delivery rate, and the maximum thrust of test articles. A nominal tube size of 1/2" has been selected for the propellant delivery, due to a reduced cost of valves and other hardware and readily available tubing, tools, fittings, etc. The size of tanks is not specified currently, as the burn time desired will vary based on the types of testing being performed.
The propellents selection is detailed more in Chapter 9, however, the two oxidisers being used, LOX and N2O, both have limitations on the safe speed of flow within a pipe. The Nitrous Oxide system has already been designed by Bentley, with a maximum mass flow rate of $2kg/s$. The maximum allowable flow speed for LOX is $15m/s$[29]. As the size of the LOX feed lines has been selected, this puts an upper limit on the mass flow rate. From this, the maximum thrust of each propellent combination can be derived.
An idealised engine model is constructed using RPA using an optimum efficiency mixture ratio, nozzle expansion to 1 atmosphere, with 100% reaction and nozzle efficiency, and no consideration of nozzle flow separation, ionisation effects, or multiphase flow. The results of these simplifications are an unrealistically efficient engine that will produce more thrust for the given propellent flow rates than any real engine, ensuring the structure is sufficiently strong for any possible engine produced for this propellent feed system.

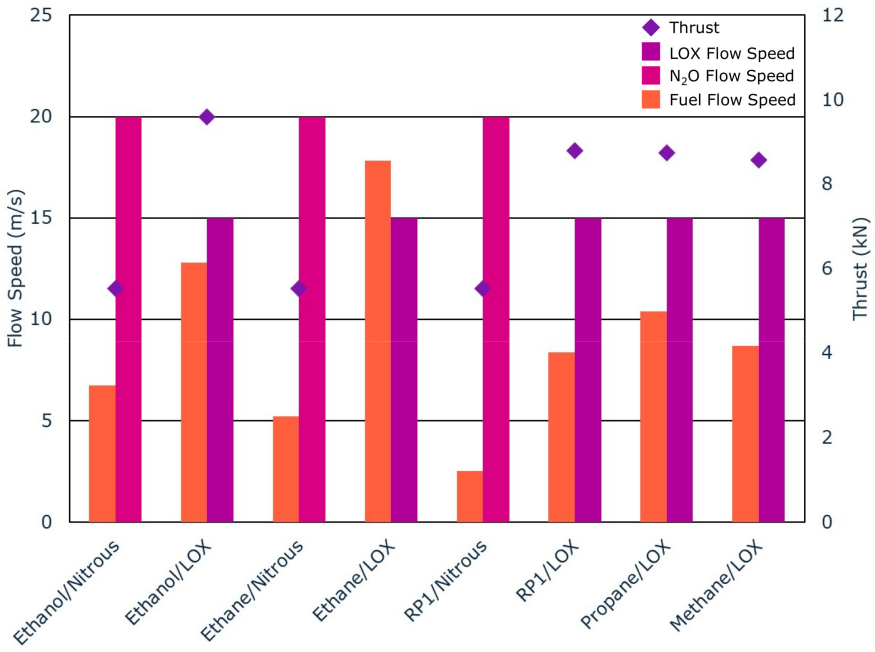



Figure 4.1: Flow speed of propellents shown in relation to thrust generated

In order to ensure all components can be adapted as test requirements change in the future, an adaptable construction is desirable. For this reason, structural steel will be used where high strength is required, as it is easy to cut and weld in the future without loss of strength, and is available at a low cost with respect to its strength. The environment at the test site is in the Peak District, and so corrosion is a concern. Stainless Steel could be used, but the increase in cost and reduction in strength do not make this a reasonable trade-off. Other weatherproofing options, such as hot dip galvanisation, would be possible and would give good results, but at an increased cost compared to readily available protective spray paints. Additionally, a spray paint coating can be easily removed as part of weld site preparation and then re-applied, allowing for easy modification if required. Parts should therefore be spray-painted when installed for a long duration at the test site. For lower strength applications, such as fluid systems, aluminium extruded sections and sheets are a low-cost option. Existing IPA and N2O feed systems use aluminium extrusions as the structural base, and this is recommended to be continued when the design of the new fluid modules is completed.

# Chapter 5

# Detailed Design

## 5.1 Thrust Frame

The Thrust Frame (TF) will be the main mechanical interface of the test article and will measure the thrust exerted. Load cells will be used for force measurement, with the frame suspended by the LCs from large concrete blocks. Alternate layouts that were investigated included the construction of A-frames, which could be directly anchored to a concrete base, or the use of a lever arm between the test article and load cell. Both of these concepts were determined to be too costly, either in terms of requiring much more steel construction or the added complexity of a bearing/bushing pivot and a lever arm section.

To mount the test article, an interface plate is secured to the thrust frame with 4 M12 bolts. The interface plate includes a 275mm diameter cutout, designed to be large enough for mounting of all engines and stages, up to and including the current largest student liquid rockets, based on an 8" tank diameter. If a larger diameter is required, a larger interface plate could be cut, and the corner supports modified.

The primary composition of the TF will be square-section steel, with cut plates and welding used to meet all of the interface requirements. Large concrete blocks are already used for various roles at the test site, so they will be used to secure all parts of the test stand. For the TF this will be in the form of concrete anchors.

The strength of the engine interface plate was found by simplifying as a circular, simply supported plate, with a force applied around a radius, with a a central hole. This approximation allows flat plate bending correlations to be used. This value was calculated using Roark's formula found in Table 11.2 case 1.e of [30], force applied to a radius with a free inner edge of a flat circular plate with a fixed outer edge. The thickness of the plate was chosen as 12mm, to

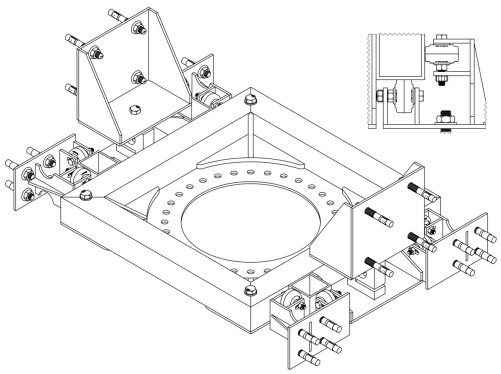


Figure 5.1: Final design for the Thrust Frame shown, with detail view of roller supports

| Failure Mode | Safety Factor | Material |
|---|---|---|
| Engine Interface Plate Bending | 2.25 | Steel S235R |
| Load Cell Bracket Weld Shear | 28.20 | Steel S235R |
| Load Cell/Anchor Bracket Bending | 2.62 (7mm) $\rightarrow$ 5.34 (10mm) | Steel S235R |
| Concrete Anchor Tearout | 47.00 | Steel S235R |
| Concrete Anchor Shear | 47.00 | Stainless Steel 316 |
| Anchor Bracket Vertical Weld Shear | 16.85 | Steel S235R |
| Interface Plate Bolt Shear | 47.00 | Stainless Steel 316 |

Table 5.1: Safety factor for several failure modes of the Thrust Frame

Figure 5.2: Concrete blocks are available on site for the construction of temporary structures

allow a reasonable safety factor without excessive thickness and weight. Maximum deflection is 0.19mm, and the maximum stress is within the yield limit. A similar simplification was used for the Load Cell and Anchor brackets, simplifying to a plate fixed on 3 sides and free on one side with a uniform distributed load. Depending on thickness, this gave varying safety factors, so any thickness between 7-10mm can be used without issue. Overall, the safety factor of each failure mode can be assessed in Table 5.1 as sufficient for safe operation. The limiting factor is the bending of brackets and flat plate components, so stiffening ribs could be added to reduce deflection if the thrust capacity is to be increased. To measure the force generated by the test article, S-Type load cells will be used. They are available in a tension and compression configuration. This is advantageous as the total weight of the test article, which could be quite high in the case of stage testing, will be suspended from the LCs at the start of the test. During a burn, a stage will present less compressive force than an equivalent engine test, so the rating of the load cell must include the range from the heaviest vehicle test possible up to the highest thrust engine test.
As most LCs available in both tension and compression have symmetric responses, twice the greater of these two figures should be selected as the total range of the LC selected. Ballast could be included on the thrust frame to reduce the risk of overloading the load cell in the case of high thrust testing. The stage mass from 5.1 was derived from the upper limit of the size

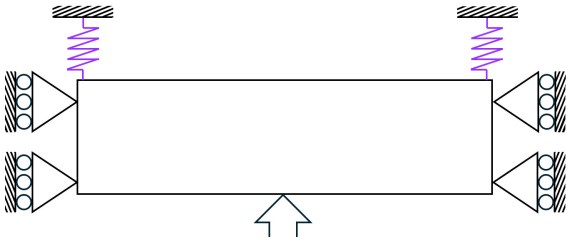


Figure 5.3: Diagram showing the support conditions of the TF, Load cells shown in purple

of amateur rockets, the legal limit within the USA[1] of $200,000\,lb\text{-}s$ ($889,600\,Ns$), defined in 14 CFR §1.1. It is assumed that the dry mass of the section to be hot fired would make up 15% of the fully fuelled mass[2]. The thrust-to-weight ratio (TWR) is included to show that any of these configurations could be feasible to fly.

| Propellant | Engine Thrust | Vehicle Mass | TWR | Burn Time |
|---|---|---|---|---|
| | N | kg | / | s |
| IPA/N2O | 5,526 | 462.2 | 1.22 | 161.0 |
| IPA/LOX | 9,587 | 375.9 | 2.60 | 92.8 |
| Ethane/N2O | 5,523 | 428.7 | 1.31 | 161.1 |
| RP1/N2O | 5,537 | 432.9 | 1.30 | 160.7 |
| RP1/LOX | 8,788 | 361.5 | 2.48 | 101.2 |
| Propane/LOX | 8,737 | 359.6 | 2.48 | 101.8 |
| Methane/LOX | 8,573 | 352.9 | 2.48 | 103.8 |

Table 5.2: Properties of full stack test per propellent

$$F_{Tension,\,Max} = g(W_{TF} + W_{stage}) = 9.81(21 + 462.2) = 4740.19N$$

$$F_{Compression,\,Max} = 9587 - (W_{TF} * g) - W_{engine} = 9380.99 - W_{engine}$$

$$F_{Total} = 29380.99 = 18,762N \rightarrow 20kN\ Rating$$

A drawback of S-Type LCs is their response to moment loading in addition to force loading. This is due to the design of the LC, however only occurs in one bending plane. The orientation used must be selected to ensure the force measurement is accurate and moments are not included, as this would reduce the accuracy of the test data. The support case of the TF can be designed such that rotation is not possible, and additionally, the orientation of the LCs can be selected such that the bending planes intersect the engine, meaning deformation of the frame is the only source of bending. In the case of a requirement to measure the magnitude of moments generated, for example, in TVC testing, it would be beneficial to include 1-2 more LCs, such that all 4 sides are supported. The orientations of the LCs would still be such that moments generated are not transferred to a bending within the LC, as the differential force can be used to determine the moment generation of the test article. For the standard two-load cell configuration, it is necessary to constrain the TF to move purely in the vertical direction without twisting. To do this, roller supports are included to be mounted alongside the LCs.

In the event of large off-axis forces, the roller supports would minimise the force applied to the LC. This reduction would reduce the risk of overloading and plastically deforming the load

[1]The USA has been used as the benchmark because there is a fixed upper limit to the impulse of an amateur rocket. Within the UK, above $10,240\,Ns$ of impulse is considered a large rocket, and any impulse is in theory possible to fly below the stratosphere (47km) if safety can be proven

[2]It is worth noting that this does not imply that a limit on the total mass of a rocket is defined here, as the recovery section, secondary stages, payloads, etc. could be removed for static fire testing

cells. Permanent damage the the cells occurs at 300% loading in the nominal direction, which would be equivalent to $\approx 35kN$ applied along a 45-degree angle to the LC. This is deemed a sufficient factor of safety as it is over 3 times higher than the highest thrust engine considered.

## 5.2 Load Cell Interface

The selected load cells must interface with the top surface of the thrust frame and be supported by concrete blocks. To ensure a secure connection, chemical or mechanical concrete anchors can be used in conjunction with the load cell interface plate. The plate was designed to meet British Standard 8539 for the specification and use of concrete anchors. For the application here, anchors should be corrosion resistant through galvanisation or stainless steel composition. The edge spacing of the holes is set according to BS 1993 at 1.2 hole diameters, and the rating of the load should take into account the preload tension and the shear loading combined. To take into account the combined loading, different approaches are required according to BS 8539, depending on the information available from the manufacturer, and this rating must be checked during the specification and procurement stage. The interface is a square pattern so that the thrust frame can be rotated 90° using the same anchors.

## 5.3 Securing Flame Deflector

For the scale model, strap eyes have been included in the design, and ratchet straps will be used to secure the deflector to a concrete block. This is deemed sufficient as the forces applied to the deflector will act into the ground and backwards into the block, so the ratchet strap should not be under high load. If in the future a more adjustable positioning system is desired, concrete anchors or a frame that attaches to the concrete blocks could be used.

# Chapter 6

# Flame Deflector Design

The flame deflector will follow the overall structure of existing deflectors: a flat impingement surface, exit curve, and uplift angle to minimise damage to the surroundings. Multiple materials and construction techniques could be used, however, the simplest is an uncooled steel deflector. The curves required could be formed through bending, casting, machining, or approximated through the use of several plates mounted at angles. Each of these approaches has drawbacks, either in cost, availability of the machines required to form the parts, or in complexity. The Rocket Test Cell (Cryogenic) at the University of Alabama in Huntsville's Johnson Research Center uses a section of pipe to divert the flow during vertical hot fires. This concept has been adapted to more closely align with existing flame deflector design guidelines while still utilising pipe sections as a readily available source of curved high-thickness steel.
Phillips outlines guidance for the exit radius and standoff distance that should be selected based on engine parameters: exit angle $\geq 2D_{exit}$, and deflector width $\geq 1.6D_{exit}$. The 5" nominal pipe size meets these requirements for the test engines, Sunfire IV and the engine detailed in [31], while being available at low cost. For static fire testing of the rocket stage, placing the engine closer to the ground reduces the difficulties of mounting and housing the test article. It is therefore desirable to minimise the distance between the bottom of the exit angle section and the impingement point by increasing the impingement angle. An impingement and uplift angle of 10°was selected, allowing a full 90°elbow pipe to be used. This is a conservative value for the uplift angle and minimises the likelihood of damage to the test site. A relatively large impingement angle will increase heating, however, a failure mode that damages the deflector is less expensive and disruptive than significant damage to the concrete or surrounding site infrastructure.

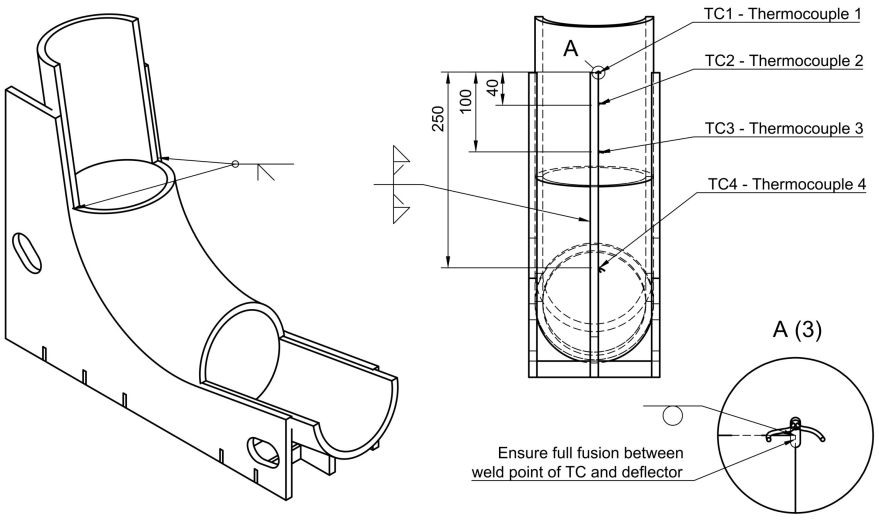



Figure 6.1: Flame Deflector design including TC mounting locations and welding instructions

To support the deflector, 3 webs have been designed and manufactured and can be seen in Figure 6.1. The central web lies directly under the impingement point and will add thermal mass to the deflector. The web will be welded with full penetration and a fillet, providing a large thermal contact area. The edge webs have a flat reference surface along the back of the FD, allowing it to be secured to a concrete block for testing. Additionally, small cross braces are welded into place to give a strong and rigid frame.
The effective pressure acting on the surface of the deflector can be determined according to [1] as $P_{ave} = \frac{T_{engine} \cdot sin(\delta)}{A_{impingement}}$ which gives a value of $430.1kPa$, . The total forces on the structure can be determined by combining the vertical and horizontal forces acting on the deflector, giving values with the higher thrust Sunfire IV of 217 N acting on the top plate, and 1741 N acting on the exit radius. The force is low on the impingement point, so support on the edges and the central support will be sufficient. For the exit radius, there will be support at the edges and uninterrupted support from the central rib. The loading into the support structure is 207 kPa from Sunfire IV, and even considering the highest thrust theoretical engine, the stress is 1.384 MPa, well below the yield strength of the steel used.

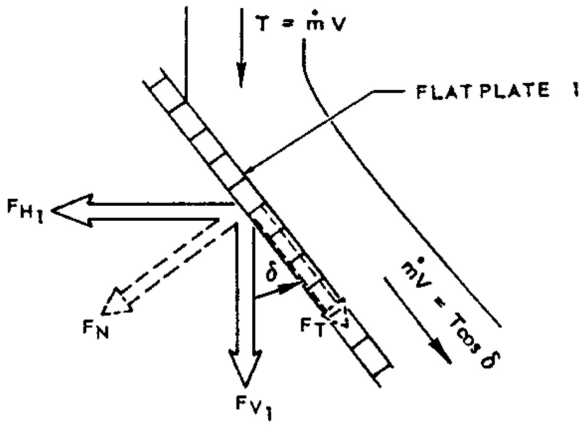



Figure 6.2: Forces acting on the impingement surface of a flame deflector, reproduced from [1]

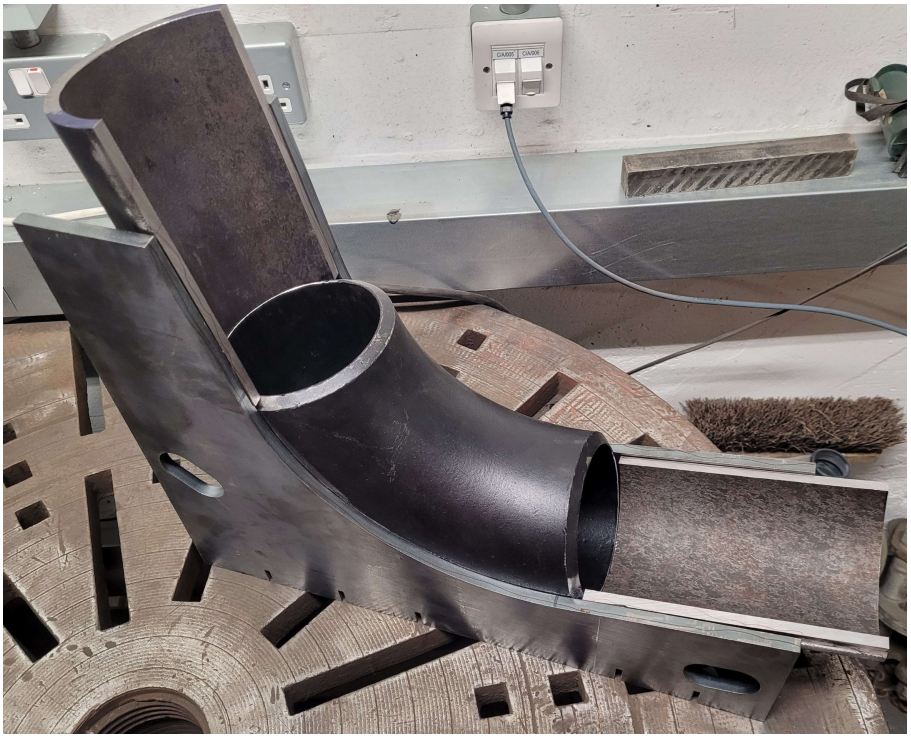


Figure 6.4: Current state of scale model flame deflector, prepared for welding

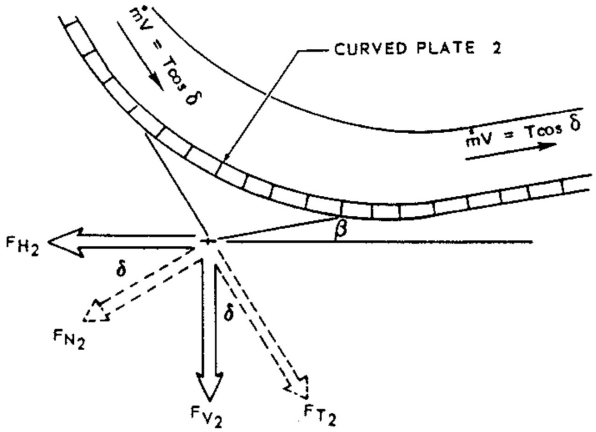



Figure 6.3: Forces acting on the exit radius of a flame deflector, reproduced from [1]

In order to reduce the size and height of the assembled deflector, a single straight section of pipe is cut to form the impingement surface and the uplift exit surface. These are welded to the 90-degree elbow section and mounted into a support frame that fixes the deflector at the desired angle. The total cost, not accounting for welding time, is under £300. Both sheet metal bending and multiple plate construction were explored, but pipe was the cheapest construction method. Currently, the deflector has been assembled and is awaiting welding to be finished, shown in figures 6.4 and 6.5.

Figure 6.5: Centre web tacked in place for welding

# Chapter 7

# Environment analysis - FLAME program

The FLow Analysis and Modelling of Engines (FLAME) program uses an analytical approach to predict the heat flux, or the surface temperature of a flame deflector when using a specified engine. The goal is to allow a full simulation to be run in <10s, with engine flow data from RPA simulations that can be quickly evaluated based on changing engine design points, leading to a quick method to compare the heat flux of an upcoming test with previous test data, and evaluate the limits of the flame deflector. This will mean that on-the-day changes are easier to accommodate, and testing of a new engine can be easily evaluated
For the first tests, an estimation of the temperature reached during a test has been produced through FEA of the deflector structure. These results have then been interpolated to allow future tests with the same deflector to have more accurate temperature estimations. The program will also implement Phillips' [1] method to predict the maximum temperature based on the duration of the burn and the heat flux. In the future, similar FEA work can be done for a larger scale deflector, though this has not been completed as a part of this project to allow test results to inform the design of the full scale FD.
The approach used combines the heat flux equations from Eckert's (eq. 7.2), Evans' (eq. 7.6), and Bartz' (eq. 7.7) methods to find the heat flux across the whole deflector, from the point of impingement to the exit of the deflector. One of the challenges of implementing these relationships was the mixture of units, and all parameters were converted to imperial as required before applying equations, and converted to metric after. Those which required a unit conversion are marked with [1].

$$q = \begin{cases} \text{Equation 7.2} & q_{Evans} > q_{Eckert} \\ \text{Equation 7.7} & q_{Evans} < q_{Bartz} \\ \text{Equation 7.6} & \text{Otherwise} \end{cases} \tag{7.1}$$

7.1: Conditions where different heat flux calculations are used

Due to Evans' solution reaching an asymtote as $x \to \infty$, the Eckert solution is used when Evans' gives a larger heat flux, as this is assumed to be due to the numerical issues encountered with a small value of $x$.

$$q_{Eckert} = (T_{wall} - T_{recovery}) St\, \rho\, C_p V_{deflector} \tag{7.2}$$

7.2: Heat flux calculated by Eckert's method

$$St = \frac{Cf}{2} Pr^{-\frac{2}{3}} \tag{7.3}$$

7.3: Eckert's Stanton number relationship

$$c_f = \begin{cases} \frac{0.0296}{Re^{0.2}} & \rho \leq 0.075 \\ \frac{0.37}{\log(Re)^{2.53}} & \text{otherwise} \end{cases} \tag{7.4}$$

7.4: Blasius and Schultz-Grunow local friction factor equation applied respectively depending on fluid density

$$T_{recovery} = \left( Pr^{\frac{1}{3}} \left( T_0 - T_{static} \right) \right) + T_{exit} \tag{7.5}$$

7.5: Ackerman [32] and Squire's [33] relationship for recovery factor in a turbulent boundary layer has been applied to find recovery temperature

$$q_{Evans} = (T_0 - T_{wall}) 0.0296 \left( \frac{\rho * V * x}{\mu} \right)^{\frac{4}{5}} \left( \frac{c_p * \mu}{k} \right)^{\frac{1}{3}} \left( \frac{k}{x} \right) \tag{7.6}$$

7.6: Heat flux calculated by Evans' method[1]

When the distance from the impingement point, $x$ is large, Evans' solution approaches Bartz'. At this point, the result from equation 7.7 is used, and the heat flux is assumed to stay at this value for the rest of the deflector surface area. The Bartz equation has been modified to remove the $\frac{A_{throat}}{A_{exit}^{0.9}}$ shape factor usually used for rocket nozzles.

$$q_{Bartz} = (T_{effective} - T_{wall}) \frac{0.026}{{D_{throat}}^{0.2}} \frac{\mu^{0.2} C_p}{Pr^{0.6}} \left( \frac{P_{chamber} g}{C*} \right)^{0.8} \sigma \tag{7.7}$$

7.7: Heat flux calculated by Bartz' method[1]

$$\sigma = \left( \frac{T_{wall}}{2T_{static}} \left( 1 + \frac{\gamma - 1}{2} Ma^2 \right) + 0.5 \right)^{-0.68} \left( 1 + \frac{\gamma - 1}{2} Ma^2 \right)^{-0.12} \tag{7.8}$$

7.8: Sigma correction factor required for Bartz' method[1]

$$T_{effective} = T_{static} \frac{1 + Pr^{0.33} \frac{\gamma - 1}{2} Ma^2}{1 + \frac{\gamma - 1}{2} Ma^2} \tag{7.9}$$

7.9: Effective temperature, required for Bartz' method[1]

These values can then be used directly as an input value for FEA analysis, compared to previous tests, or the built-in temperature estimation tools can be used to assess the risk of

melting of the deflector. These heat flux calculations require flow conditions and properties along the deflector, so a simplified fluid flow model is used, based on properties provided by the user from a separate simulation software, RPA. Several simplifying assumptions are made in both the engine modelling and the flow along the deflector. Within the engine, the expansion through the nozzle is isentropic and the flow is instantly reactive, with no further reaction in the plume, and a shifting equilibrium is used, resulting in instant property change between the chamber and exhaust.
The propagation of the flow between the outlet of the engine and the point of impingement with the deflector is modelled as presented in [2], shown in equation 7.12 At the point of impingement, an oblique shock will form as the flow is turned to meet the profile of the deflector. As this is open to the atmosphere, it is unlikely that this will take the form of a strong shock, so a weak shock solution is selected. The angles of the shock is solved numerically by finding the positive roots of the equation 7.11, a rearrangement of the $\delta$-$\beta$-$M$ equation 7.10. The flow conditions after this shock can then be found, as it is an isentropic process, and these are the flow conditions that are used as an input for the heat flux equations.
Flow along the deflector is at this point modelled as isothermal and frictionless, as the temperature, pressure, velocity, etc., after the shock are used for the full profile of the deflector. Alternate modelling choices are detailed later in this chapter.

$$tan(\delta) = \frac{2}{\tan\beta}\left(\frac{M_1^2 sin^2\beta - 1}{M_1^2\big(\gamma + cos(2\beta)\big) + 2}\right) \tag{7.10}$$

7.10: Oblique shock relationship

$$cot(\delta)^3 + \left(1\frac{\gamma+1}{2}M_1^2\right)cot(\delta)^2 + (1-M_1^2)cot(\delta) + \left[\left(\frac{\gamma-1}{2}M_1^2 tan(\beta)\right)(2+M_1^2) + (1-M_1^2)\right] = 0 \tag{7.11}$$

7.11: Cubic form of oblique shock relationship, positive roots will correspond to the value of $cot(\delta)$ where $\delta$ is the strong and weak shock angle

$$V_x = \begin{cases} V_{exit} & \text{if} \quad x < D_{ex}9.43 \\ V_{exit}\left(\frac{0.79D_{ex}}{x} - \frac{33R_x D_{exit}}{x^3}\right)^{10} & \text{if} \quad x \geq D_{exit}9.43 \end{cases} \tag{7.12}$$

## 7.1 FEA Results Interpolation

To allow quick access to temperature results during testing, the FEA results have been interpolated. These correlations can then be used to quickly estimate the temperature of each thermocouple for a certain burn time and inner surface heat flux, much faster and more simply than an FEA result or direct calculation. Implementing a true thermal model would be much more complex to implement, so this numerical approximation that maintains accuracy is used. Two FEA simulations have been conducted, the first is a flux/temperature model, where multiple inner surface heat flux values are correlated with temperature probe values. The second result used is a temperature/temperature model, where a fixed inner surface temperature is correlated to the thermal probe temperature. These are both implemented into FLAME and are used in conjunction to calculate the sensor temperatures and surface temperatures after a simulation is run. Additionally, an implementation of the Temperature/Temperature correlation allows sensor values to be input by the user to find the actual deflector surface temperature.

This is aimed at helping the evaluation of deflector performance on test days. More details and validation of the interpolation technique are included in Chapter 9.
As the FEA cases included up to a 20 second burn and $4MW/m^2$ heat flux, which would lead to extensive melting, all reasonable test cases should not exceed the reasonable limits of the interpolation. To validate the implementation of this method, the predicted values can be compared to FEA results. A test case with the following heat fluxes at the location of each temperature sensor $1.2, 0.9, 0.8, 0.5MW/m^2$ and a 12 second burn time. This is the environment that is produced by a test burn with the Sunfire IV engine, and 12 seconds is not a duration that was provided in the original results, meaning fully interpolated values will be compared to real FEA.

| | FEA | Interpolation | Error | |
|---|---|---|---|---|
| | | | °C | % |
| TS1 | 398.6 | 369.0 | 29.5 | 8.0 |
| TS2 | 284.8 | 285.6 | 0.9 | 0.3 |
| TS3 | 237.0 | 248.7 | 11.7 | 4.7 |
| TS4 | 136.6 | 156.3 | 19.7 | 12.6 |
| | | Averages: | 15.5 | 6.4 |

Table 7.1: Results from interpolated data and FEA show agreement

The fitted curves were produced with an equal heat flux across the inner surfaces of the deflector, so a non-uniform heat flux distribution can also test how large an error is introduced by non-axial temperature gradients. This test case led to an average error across the sensor locations of 6.4%, with a maximum at TS4 of 12.6% and a minimum at TS2 of 0.3%. This is equivalent to a maximum, minimum, and average error of 29.5°C (TS1), 0.9°C (TS2), and 15.5°C, respectively. This error amount is reasonable, as the goal is to estimate the point of melting of the surface of the deflector, and a prediction that was within $100C$ would cause concern and control measures should be put in place. The melting point of the ASTM A234 steel at the impingement point is $1420 - 1465C$, so an impingement point (TS1) temperature above $1300C$ should be avoided.
In order to implement this data-based method on future deflectors, the FEA analysis must be redone, with the approach taken being to set the internal surface heat flux to each value through a parametric study, and taking the probe point maximum temperature as an output. Currently ,the coefficients for the flux/temperature and temperature/temperature results of the Scale Model Deflector have been hardcoded into FLAME. As the $D$ coefficients are specific to the scale model, so the program checks that the FD matches the scale model when this method is selected. Additionally, warnings are given when the input range of the test case is exceeded. The agreement between the interpolated FEA temperature model and the analytical model presented in [1] is reasonable. There is disagreement, with the FEA model resulting in higher temperature values. In the future, assessment of the difference in experimental agreement with each model could inform a mixed approach, or modifications to either approach to improve accuracy.

| Analytical | FEA-Based | Error |
|---|---|---|
| 648.6 | 500.0 | 16% |
| 407.9 | 248.1 | 23% |
| 339.6 | 191.7 | 24% |
| 282.7 | 142.0 | 25% |

Table 7.2: Temperature results and error[1] between approaches

It is predicted that the actual model accuracy of the deflector will not be in question, with

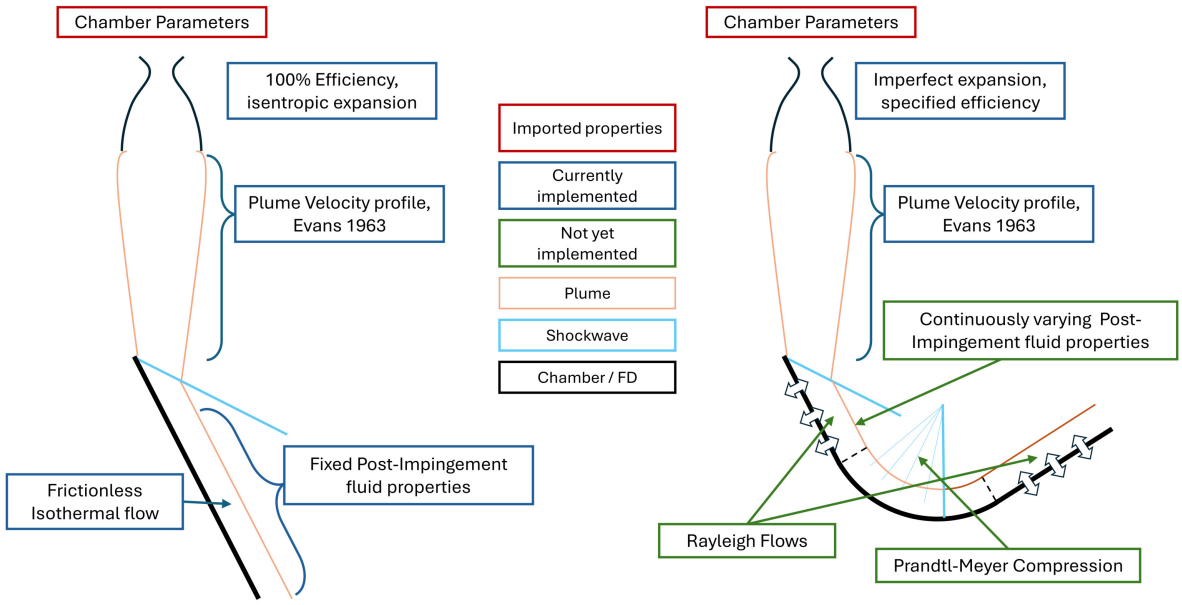



Figure 7.1: Comparison of current fluid flow model and proposed future improvements

the environment that the FD sees being the main variable, so the test will be used to verify both. Potentially, a section of deflector could be used with thermocouples mounted radially out from a test point. On this test point, a high-powered heat source could be used to heat the metal, at which point the temperature data can be compared to an FEA model of the same conditions. This test would quantify the model error and the environmental error of the hot fire tests.

## 7.2 Future Improvements

A large area of improvement would be to the user interface. The current command line interface is simplistic and does not give the user access to the engine models, flame deflector dimensions, or details of the fluid flow. The creation of a more advanced software packaging of the core heat flux calculations would make FLAME more user friendly and fully featured.

The fluid flow model could be increased in complexity to more closely model reality. The inclusion of heat flow and friction would have an impact on flow properties, and therefore on the heat transfer rate to the deflector. Heat flow could be included through the modelling of Rayleigh flows, and the curvature of the flow around the exit radius could be modelled as a Prandtl-Meyer compression. Friction could also be considered, though this would be more complex so has not been implemented.

The execution order of the code could therefore be changed to be iterative, with each iteration of heat flux altering the flow conditions. This could be repeated until the solution converges. The functions required to implement Rayleigh flows and Prandtl-Meyer compressions were created, however the iterative calculation mode was not completed as a part of this work. The proposed modified fluid flow is shown in Figure 7.1 and the code required to calculate the flow properties at each stage through Raleigh and Prandtl-Meyer flows is included in the Appendix B.

# Chapter 8

# Finite Element Analysis

## 8.1 Flame Deflector Thermal Analysis

### 8.1.1 Methodology

Thermal finite element analysis is being used to determine the temperature of the deflector under the heat flux determined by the FLAME program. To do this, the scale model FD CAD has been simplified and meshed for analysis. A full scale deflector has not been created, as there will likely be lessons learned and design changed dependent on the results of testing of the small scale model. The FEA model has been used to create two key results, the temperature lookup table function and the flux temperature function. The first is a simple model where the inner surface temperature of the deflector is set to a fixed value, and the temperature at the thermal probe locations is determined as a time dependent variable. This means that during testing, sensor values can be transformed to find the actual surface temperature of the deflector, accounting for the thermal gradient across the surface of the deflector. The flux lookup table is created by using a constant heat flux boundary condition in ANSYS, with each heat flux for a set time resulting in different temperatures at the sensor locations. Interpolation has been implemented in the FLAME program, with the 2nd degree polynomial line of best fit of each data-set generated. This is repeated for a number of simulation durations, and the coefficients are approximated for an arbitrary duration by taking another 2nd degree polynomial. Both results sets are created in the same fashion.
Model simplifications include the removal of extra supports, holes, and unnecessary features, enabling shared topology, and splitting the impingement surface into separate bodies. Shared topology is used as this makes all bodies perfectly thermally bonded, and as all parts will be welded, this is a reasonable assumption. The same material properties are used for both components for simplicity, with a value of conductivity of 51 $W/mK$ for the ASTM A106 straight section of pipe, 60 $W/mK$ for the ASTM A234 curved section of pipe, and 42.5 $W/mK$ for the S235JR support structure[1]

### 8.1.2 Mesh validation

Mesh refinement analysis was performed, and as can be seen in 8.1 increasing the fineness of the mesh beyond the $\approx 6000$ elements of the current model does not have a significant effect on the quality of the results, so this size mesh has been used for all analyses. This analysis should be repeated for models of the full scale deflector to ensure the model is of a sufficient quality.

[1]Sources: A106: www.lgpipeindia.com/astm-a106-grade-b-pipe.html; A234: www.pipingpipeline.com/1-25cr-0-5mo-1-25cr-0-5mo-si.html; S235JR: steelnavigator.ovako.com/steel-grades/s235/

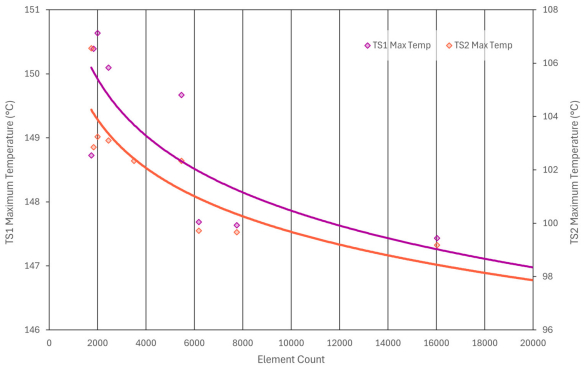



Figure 8.1: Results changing with mesh fineness, the model with ≈ 6000 elements has been selected

### 8.1.3 FEA Validation

Temperature probes have been replicated between the FEA model and thermocouple locations on the test article, and the correlation between the test data and model will be analysed. It is likely that the larger source of error will be in the estimation of the environment of the deflector, but other data, such as additional surface temperature probes, could be used to analyse the validity of the FEA model of the deflector. Alternatively a known heat flux or temperature could be used to find the temperature change over time. The factors considered in the analysis are purely the thermal input to the deflector and radiation to ambient. The natural convection effects are not included in the analysis in order to simplify the model, and this could be an additional complexity that could be validated.

### 8.1.4 Interpolation validation

In order to ensure the accuracy of the interpolation model used for the temperature of the deflector, a comparison will be made between interpolated values, an analytical approach, and FEA. A simple model of a 10mm diameter sphere was created and differing fluxes were applied to the outer surface. The conductivity was increased significantly such that the Biot number was small, and lumped analysis could be used. The temperature at a given time can be determined by each method and compared. Lumped analysis is true only when the Biot number is exactly equal to zero, implying that conductivity be infinite or external heat input is zero. Either of these cases are non-physical, but the former can be used, and becomes more accurate when increasing the conductivity of the material. A value of $1\,MW/mK$ was therefore used, with other properties of the "Structural Steel" material within ANSYS unchanged.

| Paramter | Value | Unit |
|---|---|---|
| Sphere Diameter | 10 | $mm$ |
| Density | 7850 | $kg/m^3$ |
| Conductivity | 1 | $MW/mK$ |
| Specific Heat Capacity | 434 | $J/kgK$ |
| Duration of heat flux | 18 | $s$ |

Table 8.1: Properties of the test case

$$Bi = \frac{\frac{L_c}{k}}{h^{-1}} = \frac{r_{sphere}h}{3k} = 1.66E^{-7}h << 0.01 \tag{8.1}$$

$$Q = m\,C_p\big(T(t) - T_0\big) \tag{8.2}$$

$$T(t) = \frac{qA_st}{mC_p} + T_0 \tag{8.3}$$

The training data was produced in the same way as the FD analysis, with time intervals of $5, 10, 15, 20$ seconds used. The test case was then an 18 second flux exposure, meaning that purely interpolated results were used. Disagreement between the FEA and lumped analysis aproach may be due to the FEA data being the average temperature across the body, however the very high agreement of the FEA and interpolations show that the interpolation technique can be highly accurate when compared to FEA analysis.

| | Results | | |
|---|---|---|---|
| Heat Flux | FEA | Interpolated | Analytical |
| $W/mm^2$ | $C$ | | |
| 0.1 | 339.19 | 339.19 | 337.00 |
| 0.15 | 497.78 | 497.78 | 469.09 |
| 0.2 | 656.38 | 656.38 | 618.78 |
| 0.25 | 814.97 | 814.97 | 768.48 |
| 0.3 | 973.56 | 973.56 | 918.18 |
| 0.35 | 1132.16 | 1132.16 | 1067.87 |
| 0.4 | 1290.75 | 1290.75 | 1217.57 |

Table 8.2: Average temperature found by each approach

It is predicted that the actual model accuracy of the deflector will not be in question. The environment that the FD sees is likely to be the main variable, so the test will be used to verify both. To quantify this, a section of deflector could be used with thermocouples mounted radially out from a test point. On this test point a high powered heat source could be used to heat the metal, at which point the temperature data can be compared to an FEA model of the same conditions. This test would separate the model error and the environmental error of hot fire tests.

# Chapter 9

# Fuel Choices

Private groups, individuals, and university teams have popularised the use of nitrous oxide as the oxidiser element of a bipropellent rocket. The main motivation is $N_2O$'s behaviour as a self-pressurising fluid[1]. The high vapour pressure at room temperature is sufficient to energise a pressure-fed engine cycle. This effect can be used to pressurise both propellents by the use of a moving piston. A common fuel to be used in conjunction with $N_2O$ is Isopropyl Alcohol due to high availability and reasonable performance.
Current propellants used include Isopropanol (IPA), Nitrous Oxide ($N_2O$), and Liquid Oxygen (LOX). These will therefore be included by default. Other oxidisers include Nitric Acid (RFNA) or Hydrogen Peroxide (HTP). Both of these oxidisers (RFNA, HTP) have primarily seen use in the past due to their storability, however, for student rocketry this is not a large concern, and handling and performance limitations make them not attractive options. For this reason, only $N_2O$ and LOX will be considered. The final fuels that will be considered are shown in Table 9.1.

| Fuel Name | Type | Chemical Formula | Reason |
|---|---|---|---|
| IPA | STP Liquid | $C_3H_8O$ | Already used by Sunride |
| Kerosene | STP Liquid | $C_nH_{1.953n}$ | Offers good performance with ease of handling |
| Ethane | Self-Pressurising Fluid | $C_2H_6$ | Simplifies vehicle layout while offering good performance |
| Propane | Cryogen | $C_3H_8$ | Ease of sourcing and high performance |
| Methane | | $CH_4$ | |

Table 9.1: Propellents selected with properties and justification

Additional fuels were considered, but the downsides from cost and handling difficulties were too high to justify. Liquid Hydrogen was considered, as it offers extreme performance, with specific impulse over 100 seconds higher than any other propellant combination considered. Hydrogen-oxygen combustion requires a high OF ratio, reducing the mass flow rate of hydrogen required. However, due to its incredibly low density, the volumetric requirements are very high, requiring large storage facilities for the propellant, as well as either high flow speeds or a larger diameter pipe. The first would lead to higher pressure losses and therefore increase the already higher volume of pressurisation gas required, with the latter leading to increased cost and would require new infrastructure just for hydrogen service. Both of these downsides, as well as the extreme requirements on sealing, super low temperature valves and other hardware, and therefore

[1]Self-Pressurising fluid is used in this report to mean a fluid with a high vapour pressure at room temperature. This is shown by many fluids, including LOX, however it is most useful for $N_2O$ and Ethane within rocketry due to sufficient pressure for pressure fed rocket engines as temperatures from 0-30°C

high propellant and facilities cost, lead to it being excluded from further design. Hypergolic fuels were also considered, but have been excluded mainly for safety concerns. Additionally, high cost and inability to source are key contributors.

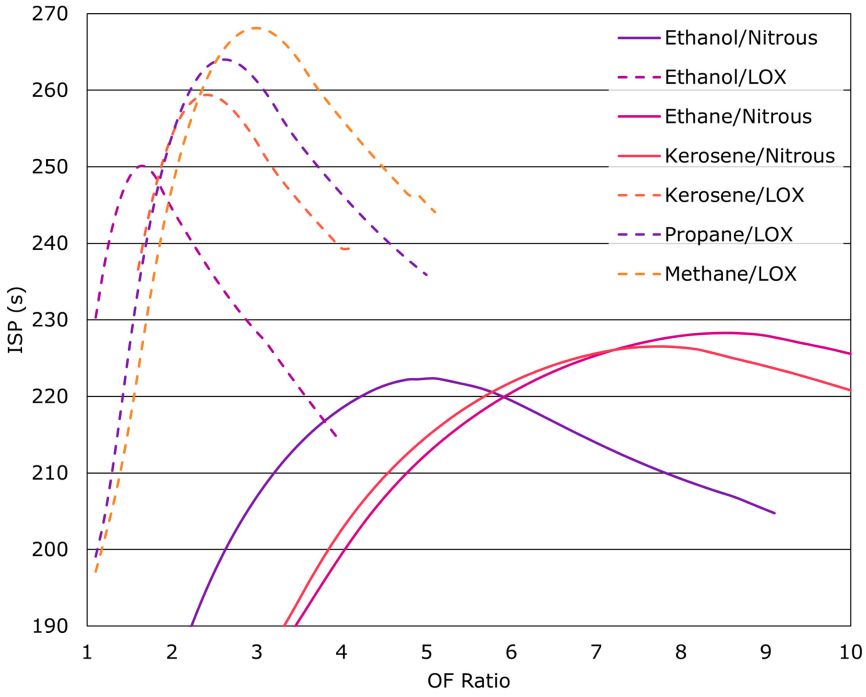



Figure 9.1: ISP of different fuel combinations, against selected OF ratio

# Chapter 10

# P&ID Design

## 10.1 Safe Design

Safe design of all modules is key. There should be no personnel within a separation distance[1] during any hot fire or cold flow operation; however, including aspects of safe design will still minimise risk. During filling and some drain procedures, there may be a possibility for pressurised sections of piping while operators are near to equipment, and for this reason, there must be testing in place to ensure the assembled fluid system can safely operate. Principles that should be followed include the use of pressure relief valves and burst disks to prevent a boiling liquid expanding vapour explosion (BLEVE). Additionally, pressure gauges should be included to ensure operators know when a section of pipe is de-pressurised, and therefore safe to remove fittings, etc. Some fluids have additional requirements for flow control valves, manual dump valves, etc., and these must be included.

In the case that fluid pipes or pneumatic gas lines are filled while operators are working on the fluid panels, there is a risk of an oxygen deficiency hazard (ODH), which will produce an irrespirable atmosphere. ODH can be mitigated through suitable environments and ventilation, working outside, training operators on the risks of ODH and how to spot it, and the use of oxygen monitors. Portable oxygen monitors could be used to monitor the presence of asphyxiating or toxic fumes in the air and ensure safety for operators. Some gases would pose a hazard due to producing an asphyxiating atmosphere, such as nitrogen. Other substances could pose an imminent hazard to health regardless of oxygen content in the atmosphere, and this must be evaluated to ensure that the concentration of these substances is within workplace exposure limits, and immediately dangerous to life or health (IDLH) airborne concentrations are not reached. Enriched levels of atmospheric oxygen can also lead to hazards, as can exposure to liquid oxygen. Natural fibres should be worn where possible to reduce the severity of potential burns. Additionally, hydrocarbons contained in asphalt, rubbers, etc., can have an increased risk of combustion, so contact must be minimised. Mitigations of these hazards must be assessed through a sufficient and suitable risk assessment to ensure that work can be completed safely.

A key failure mode to be assessed is a failure of ignition in the combustion chamber. In the case that the fuel and oxidiser are ejected from the chamber without ignition, they could pool, and in the case of cryogens, form a combustible atmosphere from the mixture of oxygen and a fuel. Nitrous oxide would likely not lead to the production of oxygen, as the decomposition process would not start, however, the anaesthetic effects could pose a risk to operators. The mitigation of this risk will likely be ventilation, waiting for the gases to dissipate, but assessment

[1] The separation distance can be calculated according to US 14 CFR §420.69 or another method according to site regulations

should be completed on site, and use of a pilot flame or similar could be an alternative mitigation. The solution chosen must be allowable by the Dangerous Substances and Explosive Atmospheres Regulation 2002 (DSEAR), as with the whole fluid system.
All high-pressure piping should be leak tested, which could include but is not limited to: snoop testing at a low pressure (0.5-1 Bar), full operating pressure leak tests, or hydrostatic tests.

### 10.1.1 Cryogens

Designing for cryogens faces a few key considerations due to the extreme operating temperature and the potential of the liquefied gases to expand hundreds of times when allowed to fully phase transition to a gas. Temperatures as low as 90K lead to issues in handling, operations, and the strength of key components. Materials must be selected that maintain their strength at lower temperatures, insulation must be used to prevent the formation of ice and to offer protection to persons working on the system while it is still cold. Additionally, all vents must be designed to mitigate the risk that the formation of ice would impede their function, especially for safety-critical operations.
Operationally, it must be ensured that there is no trapped volume of liquid gas, as this will cause a BLEVE. This can be mitigated by ensuring all pipes are vented to the atmosphere, letting the gas escape.

### 10.1.2 Supercritical fluids

The critical point for nitrogen is $-147C$ at 34 Bar. As this is within the operational envelope of the LOX module, supercritical effects must be accounted for. Additionally, the critical points for $N_2O$ and Ethane are attainable temperatures for a hot day, $32.2C$ and $36.5C$ respectively. While this would be an unusually warm day, especially at Buxton, it's within the realm of possibility, so all hardware must be designed with the possibility of supercritical flow in mind. When in a supercritical state, the fluid becomes less predictable, with the interlink of temperature, pressure, and density leading to non-linear, dynamic behaviour.

### 10.1.3 Self-Pressurising Fluids

Nitrous Oxide and Ethane have a high vapour pressure at room temperature, and as such will boil at room temperature. When contained, they will build pressure up to a point where equilibrium is met. The temperature of the fluid when stored is therefore a critical characteristic, as it can be used to determine the pressure the fluid is at. When the temperature increases a small amount, this can lead to a large increase in pressure, so care must be taken. The pressure that $N_2O$ and Ethane reach is not sufficient to risk bursting pipes, however, unexpected high-pressure volumes could cause issues during operations and pose a risk to operators, so trapped volumes must be avoided due to the same BLEVE risk.

## 10.2 Design Principles

### 10.2.1 Modularity

Wherever possible, design work should not be repeated. This principle will allow more design work to be completed, and additionally means that designs of support hardware, pipe routing, etc., can be reused. Some key modular sections include:

- the interface from the pressurisation inlet to the run tank accumulator,
- the interface system used for the inlets and outlets of modules,
- the propellant dump tanks, and

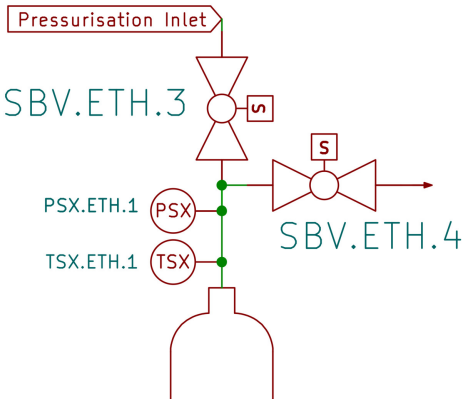



Figure 10.1: P&ID for the supercharger module

- the piping used to fill the run tanks when cryogenic or pressurised liquids are loaded.

Each of these subsystems will be designed, with modules utilising these modules wherever possible. Due to each propellent having different key safety requirements, each module should be treated as its own design, with modifications or changes to one not necessarily being carried over to another without the new design being properly considered. Additionally, contamination is a risk for parts that will see high-pressure oxygen service, so modules should not be swapped between modules without a full deep clean. All remote-actuated valves have been labelled as SBV - Servo Ball Valve. Currently, this is the preferred type of valve; however, in the future, switching to pneumatically actuated flow control valves (FCV's) would likely be faster acting and more reliable. These can be switched out without requiring other design changes, other than routing a nitrogen line, which can be fed off of an additional regulator from the high-pressure nitrogen bottle.

### Supercharger Module

The supercharger module, shown in Figure 10.1, is named after the method to increase the consistency of the pressure delivery when additional pressurising nitrogen is used, when compared to a purely self-pressurising system. This sub-module provides pressure readings of the run tank and includes valves for the pressurisation and depressurisation of the run tank. For self-pressurising and cryogenic systems, a temperature sensor will also be included to monitor the natural vapour pressure before nitrogen is introduced, and to understand thermal differences between the ullage volume and the delivery properties of the propellant.

### Dump Module (Cryo)

The dump sub-module for cryogenic systems must function to remove excess or leftover propellant from the system in a safe fashion, when it is too contaminated or otherwise it is not possible to move the fluid back to the long-term storage tank. These propellents could pose a hazard due to cold burns or asphyxiation, and this hazard must be mitigated through protective enclosures, adequate ventilation, etc. Crucially, the dumping of a cryogenic fuel and liquid oxygen, and the potential mixing of gaseous fuel and oxidiser, must be evaluated for the hazard posed. Ventilation systems may have to be fitted that vent the gases away from each other. The final requirements are that the valves are not at risk of ice formation that may impede their operation, and that actuation is easy through the use of an electronic actuation switch that is readily accessible to operators, or by the addition of a secondary manual dump valve. To ensure

the safety of this critical component, as it will form a key part of emergency procedures, a full risk assessment must be performed.

### Dump Module (Hazardous)

For hazardous or environmentally harmful fluids, it is not permissible to release the material into the environment. The system must therefore contain the fluids and their fumes to a practical level, from being released. The container into which the fluid is moved must be chemically resistant to the fluid in question and have some amount of spill protection.

### Fill Module (Pressurised & Cooled)

The filling interface should be readily accessible without a need to lift or bend unnecessarily, such that gas bottles or other standard containers can be connected with a reasonable length of flexible hose. There should be valves included to allow the start and stop of fluid flow, to open the run tank to atmosphere when desired, and some mechanism to prevent siphon effects. The same fill piping is used for the LOX, Ethane, Nitrous Oxide, and Liquid Methane and Propane modules.

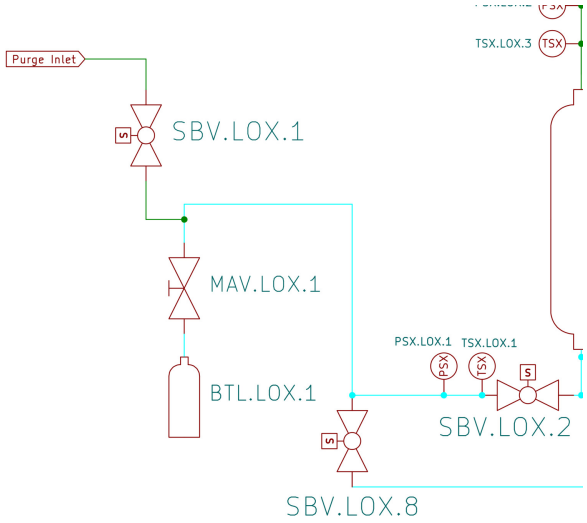



Figure 10.2: Filling system for the LOX P&ID

# Chapter 11

# Sample Operations

The sequencing of valves used will depend heavily on the working fluid that is being used. For cryogens this process will include a cool-down sequence to ensure fluid can flow fully to the test article without additional boil-off. Additional considerations are to the change in phase transition temperature with pressure, and the changes in the properties of the pressurant gas at certain temperatures and pressures. To ensure cold, dense, uncontaminated LOX is provided, the tank should be opened to atmosphere shortly before the test begins, reducing the boiling temperature of the LOX and utilising the latent heat of vaporisation to reduce the temperature of the liquid phase. As nitrogen is introduced to the tank, it will cool. The room pressure condensation temperature of nitrogen is lower than the liquid temperature of the oxygen, however as pressure builds and the triple point of the nitrogen is exceeded, a layer of supercritical liquid nitrogen will begin to form at the interface [34] when pressures are above 34 Bar. To minimise contamination by diffusion of nitrogen into the $O_2$-$N_2$ solution, the introduction of nitrogen should be delayed as late as possible. An addition way to reduce the contamination of LOX is to ensure the area of contact is minimised, by reducing the diameter of the tank.

## 11.1 Liquid Oxygen chill-down and run

The liquid oxygen module must be operated with care to reduce the risk of trapped volumes of liquid and maximise performance. Sample operations are included here in the format that is suggested for future use. Each step has a valve or other instrument to use, an action to take, and sections for additional notes or settings. This ensures that each instruction is clear and each line can be checked off. This could in the future be replaced with an automated system, where instructions can be imported and performed autonomously.

| Instrument | Action | Setting |
|---|---|---|
| SBV.LOX.1 | CHECK | CLOSED |
| MAV.LOX.1 | OPEN | |
| TSX.LOX.1 | WAIT UNTIL | <100 K |
| SBV.LOX.3 | CHECK | CLOSED |
| SBV.LOX.2 | OPEN | |
| SBV.LOX.4 | OPEN | |
| TSX.LOX.2 | CHECK | $\approx$ 90 K |
| TSX.LOX.3 | WAIT UNTIL | $\approx$ 90 K |
| SBV.LOX.2 | CLOSE | |

Table 11.1: LOX system run tank fill operation

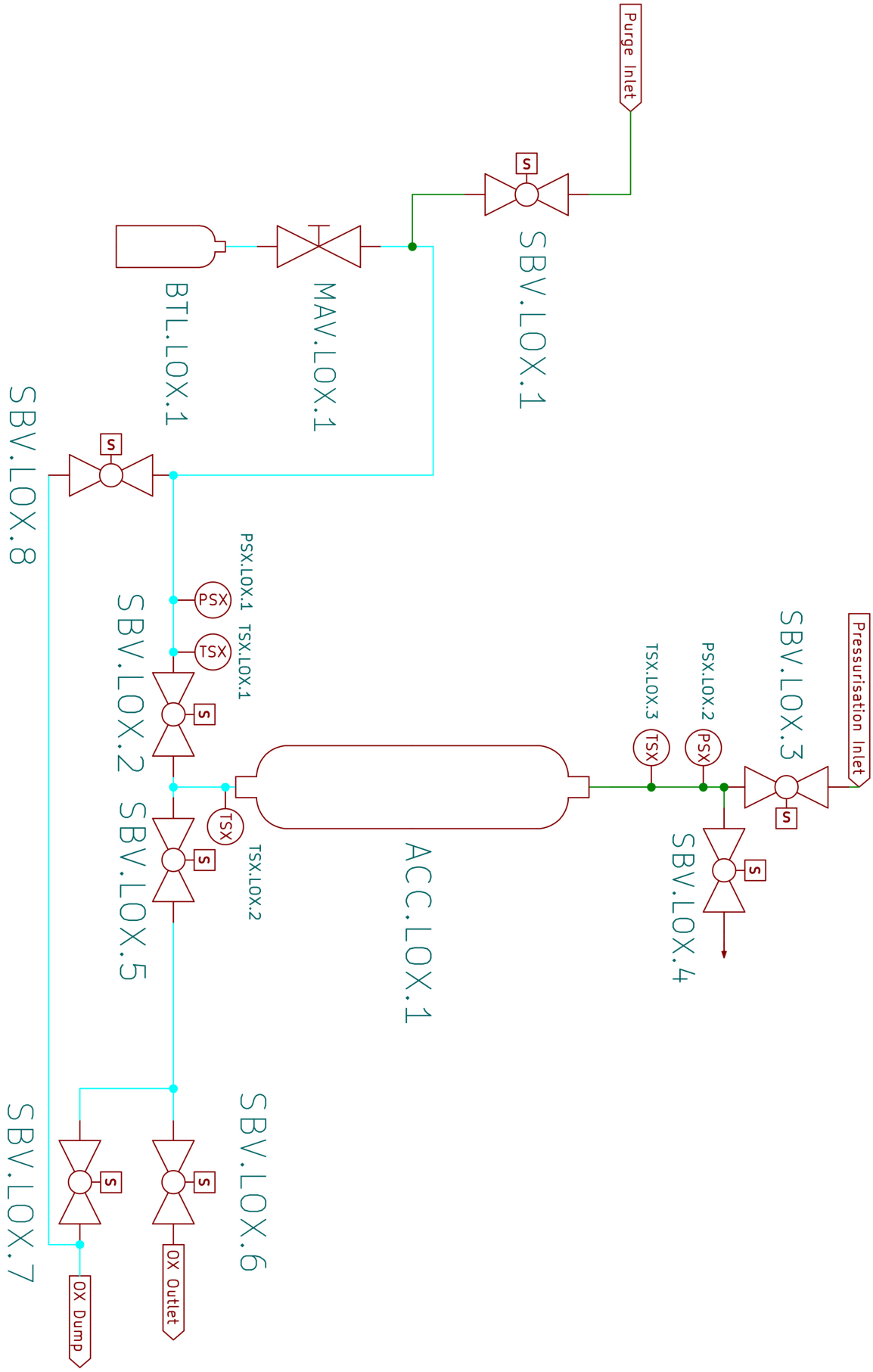



Figure 11.1: LOX Panel P&ID

| Instrument | Action | Setting |
|---|---|---|
| SBV.LOX.5 | OPEN | |
| SBV.OXI.2 | OPEN | |
| TSX.OXI.1 | WAIT UNTIL | $\approx$ 90 K |
| | WAIT FOR | 10S |
| SBV.OXI.2 | CLOSE | |
| TSX.OXI.1 | MONITOR | |
| TSX.LOX.2 | MONITOR | |

Table 11.2: LOX feed line chill down operation

| Instrument | Action | Setting |
|---|---|---|
| SBV.LOX.4 | OPEN | |
| SBV.LOX.6 | OPEN | |
| TSX.LOX.2 | WAIT UNTIL | 90K |
| SBV.LOX.4 | CLOSE | |
| SBV.GN2.6 | OPEN | |
| PSX.LOX.2 | WAIT UNTIL | Desired Pressure |
| SBV.GN2.6 | CLOSE | |
| SBV.OXI.1 | OPEN | |

Table 11.3: LOX feed operation

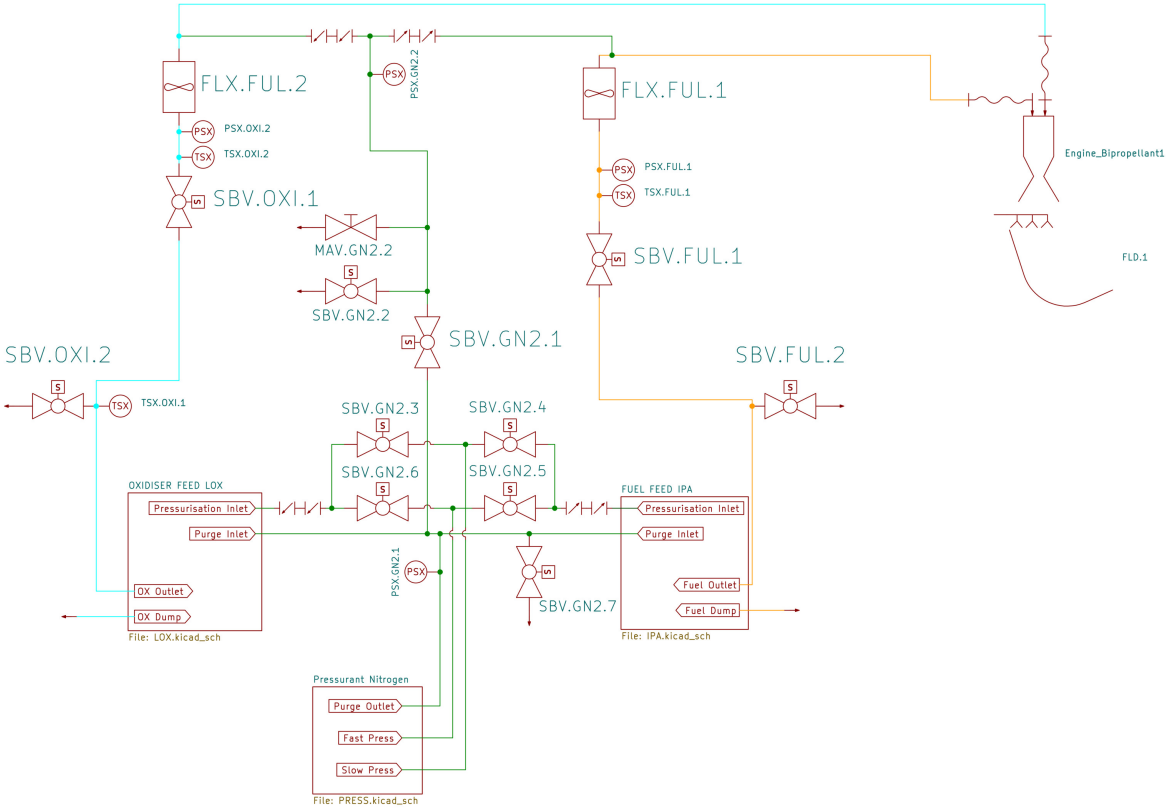



Figure 11.2: Full Test site P&ID

# Chapter 12

# Discussion

## 12.1 P&ID Designs

The piping and instrumentation layout for all propellents has been completed as planned, and can be assembled and used as required. Unfortunately, more detailed mechanical design of the most likely modules to be constructed within the next few years - LOX, IPA and Kerosene - has not been completed. This is not anticipated to be a large obstacle to future tests, as the kerosene and IPA modules will share many design elements due to the similar P&ID layout, and the LOX module can therefore be the focus of future work.
The sample operation included could be used as a baseline, and as more testing is undertaken, the operational planning processes used within Sunride will mature. Some considerations of LOX and cryogen operations have been noted, but full safety analysis and testing will have to take place to commission any new fluid feed system installed at Buxton.

## 12.2 Structural Design

Technical drawings have been produced that detail the necessary steps to manufacture the hardware required to conduct testing with a thrust up to 10kN. The design produced can be manufactured from commonly available sheet metal and bar stock, aiding the accessibility of the design. One key aspect that is not considered in detail is the strength of the concrete anchors, as this is dependent on the composition and layout of concrete blocks used. Future work must ensure that both the tear-out forces and shear strength of the anchors is considered, and BS 8539 is followed.

## 12.3 FLAME

The FLAME program implements relationships from the literature to quickly solve the problem of plume impingement heat flux. In the future, both the software and fundamental computation could be improved through the addition of a GUI or experimental investigation into hot gas flows and shock attachment heat flux. Currently, the tool will be useful for hot fire tests that utilise the small-scale deflector, and will soon be verified during hot fire tests. After this test validation is completed, FLAME can be used to rapidly assess the variable operating point of an engine, and simplify test operations. After initial hot fire testing, design and construction of a full-scale deflector based on a 10” nominal pipe diameter can be constructed. FEA of this larger model can be completed, and FLAME can be updated to include this FD. At this point, testing can continue and the software should be suitable to aid testing of all potential engines regardless of propellent used or thrust level.
The FEA interpolation implementation was an additional goal that was added to the soft-

ware, and is much simpler to implement than an analytical or numerical solution. Interpolation is likely a better option that full temperature modelling, due to computational simplicity, so adding more complex computation may not be necessary. The conditions used in the FEA modelling can be more complex; in this case, radiative heat transfer to the ambient surroundings was included. In the future, more advanced modelling, including free convection, could be completed and implemented without altering current programming.

There was an initial investigation into modelling the flow across the deflector in more detail, including Rayleigh flow effects and the flow around the deflector being modelled as a Prandtl-Meyer compression. These were implemented; however, including this in the flux calculation model was too complex to be implemented within the last months before this report was submitted.

An additional point of potential error is the calculation of the impingement heat flux. The calculation method used relies heavily on the coefficient of friction calculation, which has several available approximations. Two calculation methods were used, shown in Equation 12.1, with plume properties determining which to use, such that the values determined by Evans' and Eckerts' relationships were comparable. A better relationship could be developed by replicating the work in [35] experimentally or numerically. Schlutz-grunow models cold flows, with a stagnation temperature under $100°F$ (37.8°C), much lower than the values of the flow of interest here. Investigating how the relationship changes could inform changes to the $C_f$ calculation within FLAME. Alternatively, the actual heat flux at the impingement point could be found experimentally and compared to the predicted results, investigating if other sources of error dominate.

$$C_f = \begin{cases} \frac{0.0295}{Re^{0.2}} & \text{if } \rho < 0.08 \\ \frac{0.37}{\log(Re)^{2.53}} & \text{if } \rho \geq 0.08 \end{cases} \tag{12.1}$$

# Chapter 13

# Conclusions

This project has delivered a sample flame deflector which enables vertical hot fire testing of the upcoming Black Adder rocket. These tests can be used as a key data point in the verification of FLAME and the thermal FEA model of the FD. Other key deliveries include detailed technical drawings that would enable high thrust testing, and fluid feed system plans for alternate fuels that can be used for a variety of future testing. More work is needed both on the detailed design of these systems and on validating the results of FLAME. This is content that could be conducted by other Sunride members, or as a part of an FYP or similar. The work of Bentley has been incorporated in the form of the $N_2O$ feed panel being reused.
The overarching goal to provide a base of knowledge for possible future expansion to the Buxton test site has been met, and future work can build on the designs here to allow continued expansion of the scope of Sunride projects.

## 13.1 Self Reflection

I have enjoyed the challenges this project has posed, as the wide variety of types of content required: mechanical designs, fluid flow diagrams, safety analysis, propulsion analysis, FEA, etc., have kept the work new and exciting throughout the project. I am particularly proud of the work on FLAME, the scope was allowed to grow as the usefulness of the software became clear, and I hope that it can be improved and maintained in the future. Certain goals of the project were reduced in scope: detailed fluid system, mechanical design, and more in-depth operational planning.
Due to timeline limitations on hot fire testing at Buxton, experimental validation has not been completed, though this was out of my control. I am confident that the designs will be able to withstand the plume environment, and I hope that the agreement between experiment and modelling is high. The shift in balance of this report from more design-focused to including more analysis has been an effective choice, and has made more useful contributions to Sunride's propulsion work. Overall, the project aims have been met, and in some cases exceeded, and key analyses have been completed with an increased scope. The combination of three heat flux calculation methods gives more assurance in the quality of the result, as agreement is reached. Additionally, an effective and usable software program and accompanying designs have been delivered. I am confident that these designs can form a baseline and continue to be useful as the test scope expands.

# Bibliography


[1] J. D. Phillips, "Flame Deflector Design, Standard For," 1990.

[2] R. L. Evans and L. Sparks, "LAUNCH DEFLECTOR DESIGN CRITERIA AND THEIR APPLICATION TO THE SATURN C-1 DEFLECTOR," 1963.

[3] A. Bentley, *Propellant Delivery Systems for a Small-Scale Rocket Engine Test Rig (C.H.I.M.E.R.A)*. PhD thesis, University of Sheffield, May 2024.

[4] M. R. Dudley, "Experimental Techniques for Three-Axes Load Cells Used at the National Full-Scale Aerodynamics Complex," 1985.

[5] L. M. Calle, P. E. Hintze, C. R. Parlier, B. E. Coffman, J. W. Sampson, M. R. Kolody, J. P. Curran, S. A. Perusich, D. Trejo, M. C. Whitten, and J. Zidek, "Refractory Materials for Flame Deflector Protection System Corrosion Control: Similar Industries and/or Launch Facilities Survey," 2009.

[6] W. M. Dziedzic, "FLAME DEFLECTOR ABLATION ANALYSIS BASED ON ARTEMIS I LAUNCH ENVIRONMENT," Aug. 2023.

[7] Z. Zhou, Y. Bao, P. Sun, and Y. Li, "Cooling of rocket plume using aqueous jets during launching," *Engineering Applications of Computational Fluid Mechanics*, vol. 16, pp. 20–35, Dec. 2022.

[8] Y. Jiang, S. Yu, J. Li, and F. Zhou, "Cooling Effect of Water Injection on a High-Temperature Supersonic Jet," *Energies*, vol. 8, pp. 13194–13210, Nov. 2015.

[9] T. Luo, K. Wang, and M. Yao, "Effect of water injection on the cooling performance of flame deflector in rocket engine testing platform," *Applied Thermal Engineering*, vol. 236, p. 121687, Jan. 2024.

[10] S.-I. Kang, J.-W. Nam, and H.-I. Huh, "A Study for Rocket Exhaust Flow Cooling due to the Central Spray Type Water Injection," 2013.

[11] W.-S. Seol and Y. Moon, "A Study of Core Water Injection Effect Influencing Plume in 75 tf 1 st Stage Liquid Propellant Rocket Engine Ground Test," Apr. 2015.

[12] A. Sharma, T. J. Tharakan, and S. S. Kumar, "Analysis for design optimization of high thrust liquid engine hot test facility," *Acta Astronautica*, vol. 193, pp. 653–666, Apr. 2022.

[13] K. Yang, Y. Qiang, C. Zhong, and S. Yu, "Study on the Effect of water Injection Momentum on the Cooling Effect of Rocket Engine Exhaust Plume," *J. Phys.: Conf. Ser.*, vol. 916, p. 012047, Oct. 2017.

[14] J. Yi, M. Yanli, W. Weichen, and S. Liwu, "Inhibition Effect of Water Injection on Afterburning of Rocket Motor Exhaust Plume," *Chinese Journal of Aeronautics*, vol. 23, pp. 653–659, Dec. 2010.

[15] D. Fu, Y. Yu, and Q. Niu, "Simulation of underexpanded supersonic jet flows with chemical reactions," *Chinese Journal of Aeronautics*, vol. 27, pp. 505–513, June 2014.

[16] C. Zhao, Y. Su, Z. Sun, Y. Wang, and G. Le, "Drift and afterburning effect on rocket exhaust plume impingement and optimization of deflector," *Advances in Space Research*, vol. 73, pp. 3963–3975, Apr. 2024.

[17] L. Wang, Q. Cheng, J. Wang, Y. Zhang, W. Zhang, S. Liu, and Z. Xia, "The effect of secondary boundary layer combustion of hydrogen on rocket plume heat release characteristics," *International Journal of Hydrogen Energy*, vol. 71, pp. 1174–1190, June 2024.

[18] J. K. Ignatius, S. Sathiyavageeswaran, and S. R. Chakravarthy, "Hot-Flow Simulation of Aeroacoustics and Suppression by Water Injection During Rocket Liftoff," *AIAA Journal*, vol. 53, pp. 235–245, Jan. 2015.

[19] L. Huang and M. S. El-Genk, "Heat transfer of an impinging jet on a flat surface," *International Journal of Heat and Mass Transfer*, vol. 37, pp. 1915–1923, Sept. 1994.

[20] A. Radzman and N. Uddin, "Numerical simulation of supersonic jet impingement," in *7th Brunei International Conference on Engineering and Technology 2018 (BICET 2018)*, pp. 1–4, Nov. 2018.

[21] C. Lu, Z. Zhou, X. Liang, and G. Le, "Thermal Environment of Launch Pads During Rocket Launching | Journal of Spacecraft and Rockets," 2022.

[22] H. Cai, W. Nie, K. Guo, and S. Zhou, "Influence of deflector on impact properties of multi-nozzle LOX/kerosene engine exhaust plume," in *2017 8th International Conference on Mechanical and Aerospace Engineering (ICMAE)*, pp. 296–300, July 2017.

[23] S. Sandeep, M. Sharma, and P. Krishna, "Designing jet deflector configuration for a semi-cryogenic rocket engine," *Physics of Fluids*, vol. 35, p. 116104, Nov. 2023.

[24] R. Prickett and E. Mayer, "Rocket Plume Impingement Heat Transfer on Plane Surfaces," *JSR*, vol. 24, pp. 291–295, Aug. 1987.

[25] E. T. Piesik, R. R. Koppang, and D. J. Simkin, "Rocket-exhaust impingement on a flat plate at high vacuum," *Journal of spacecraft and rockets*, vol. 3, no. 11, pp. 1650–1657, 1966.

[26] D. Bartz, "Turbulent Boundary-Layer Heat Transfer from Rapidly Accelerating Flow of Rocket Combustion Gases and of Heated Air," in *Advances in Heat Transfer*, vol. 2, pp. 1–108, Elsevier, 1965.

[27] A. Ponomarenko, "Thermal Analysis of Thrust Chambers," 2012.

[28] E. R. G. Eckert, "Engineering Relations for Heat Transfer and Friction in High-Velocity Laminar and Turbulent Boundary-Layer Flow Over Surfaces With Constant Pressure and Temperature," *Transactions of the American Society of Mechanical Engineers*, vol. 78, pp. 1273–1283, Aug. 1956.

[29] E. I. G. A. A. , "Doc. 13/20 Oxygen Pipelines and Piping Systems," 2012.

[30] W. C. Young and R. G. Budynas, *Roark's Formulas for Stress and Strain*. McGraw Hill LLC, Oct. 2001. Google-Books-ID: pummClLoFXEC.

[31] R. Hurcombe, *Liquid Rocket Engine Design*. PhD thesis, University of Sheffield, May 2024.

[32] G. Ackerman, "Plate thermometer in high velocity flow with turbulent boundary layer," *Forshcung auf dem Gebiete des Ingenieurwesens*, vol. 13, 1942.

[33] H. B. Squire, "Heat transfer calculations for aerofoils," tech. rep., 1986.

[34] A. J. Zuckerwar, T. K. King, and K. C. Ngo, "Contamination of liquid oxygen by pressurized gaseous nitrogen," Apr. 1989. NTRS Author Affiliations: Old Dominion Univ., NASA Langley Research Center NTRS Report/Patent Number: L-16526 NTRS Document ID: 19890010128 NTRS Research Center: Legacy CDMS (CDMS).

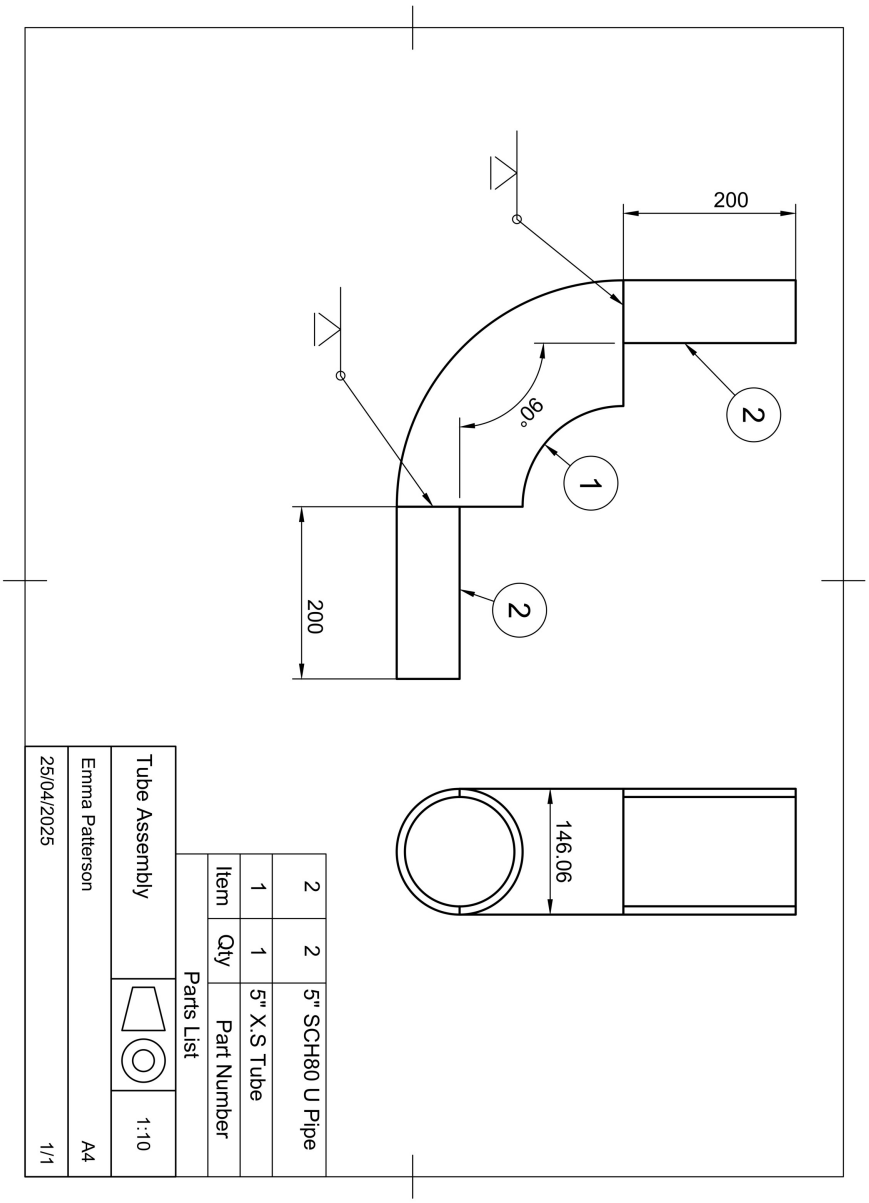
[35] F. Schultz-Grunow, "New frictional resistance law for smooth plates," Sept. 1941. NTRS Author Affiliations: NTRS Report/Patent Number: NACA-TM-986 NTRS Document ID: 19930094430 NTRS Research Center: Legacy CDMS (CDMS).

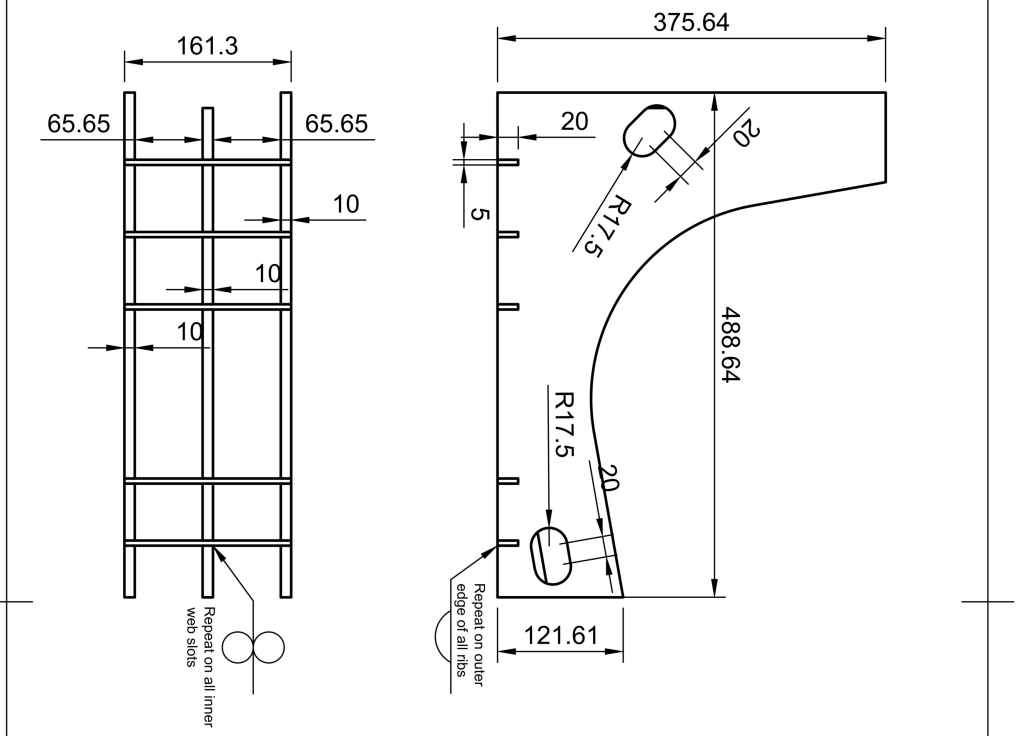
# Appendix A

# Final Design Documents

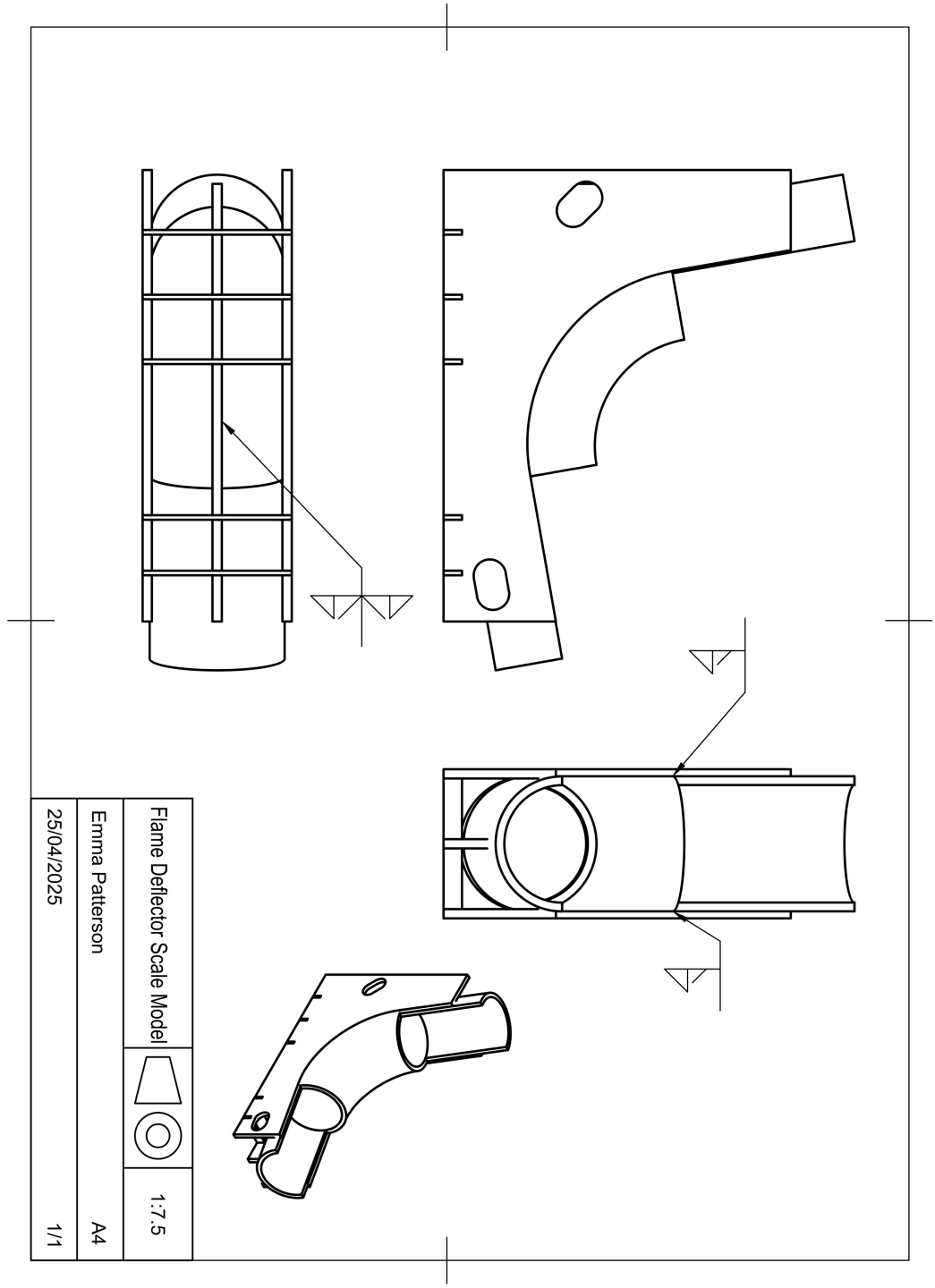
## A.1 Technical Drawings

### A.1.1 Flame Deflector

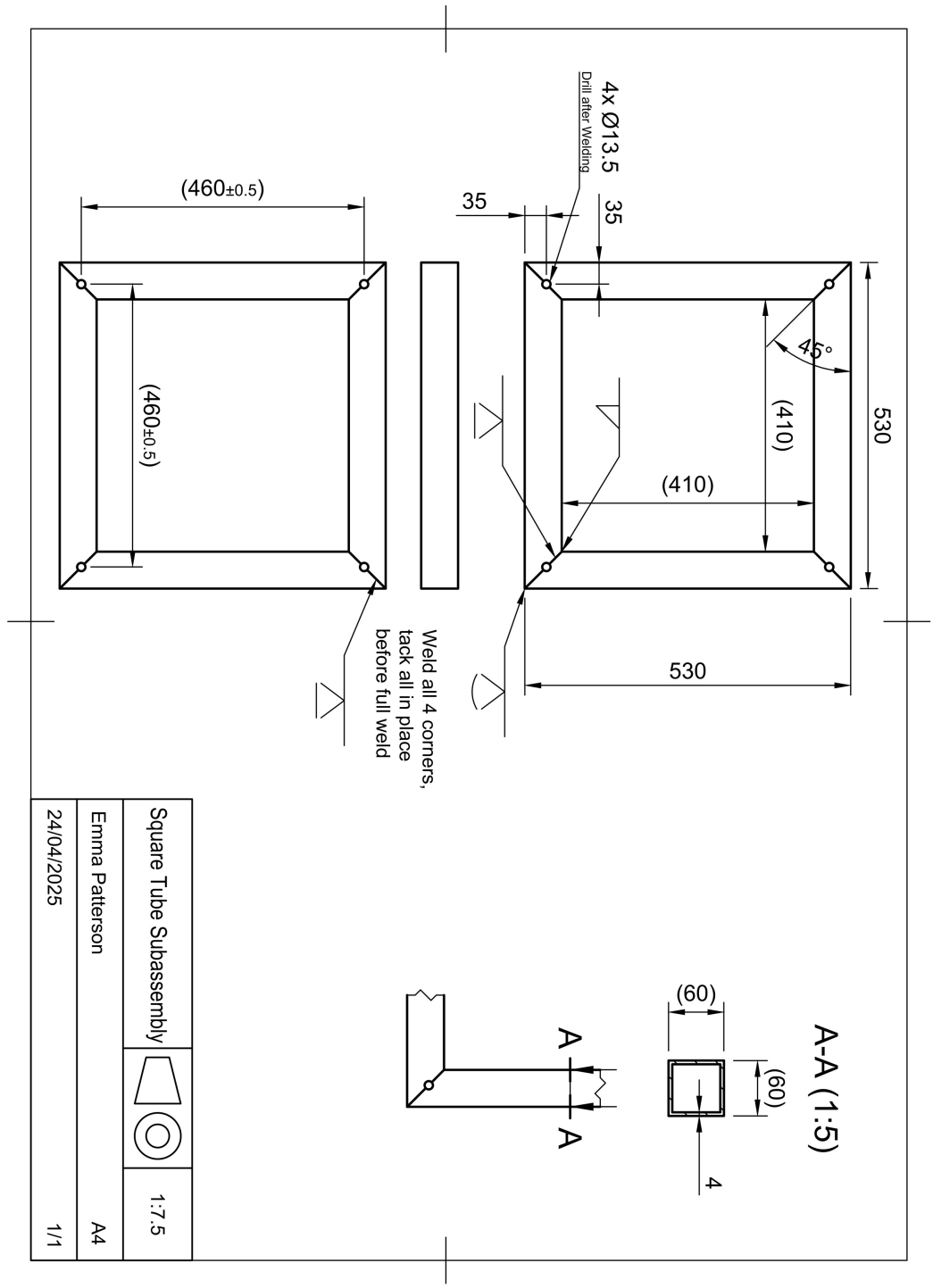


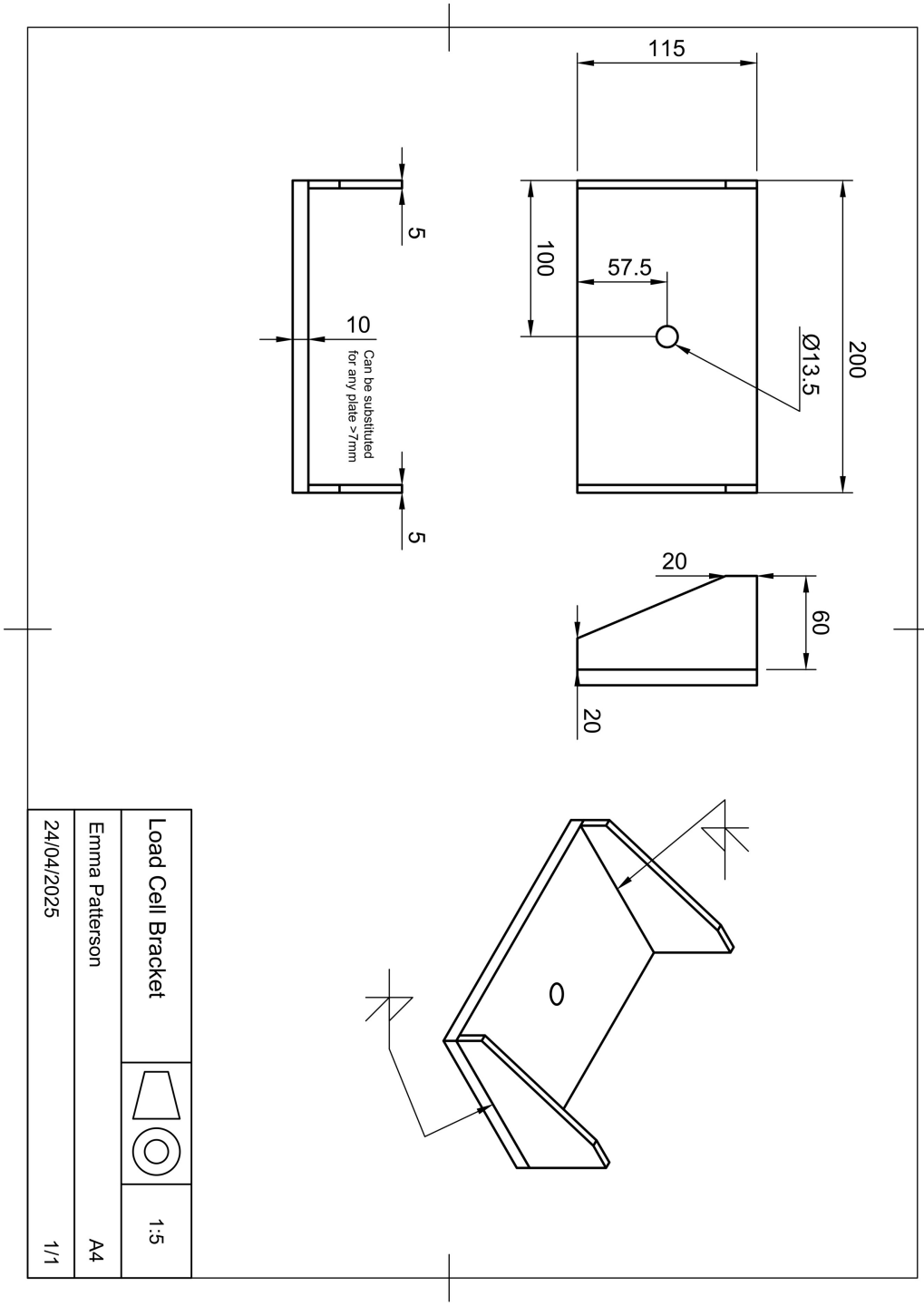


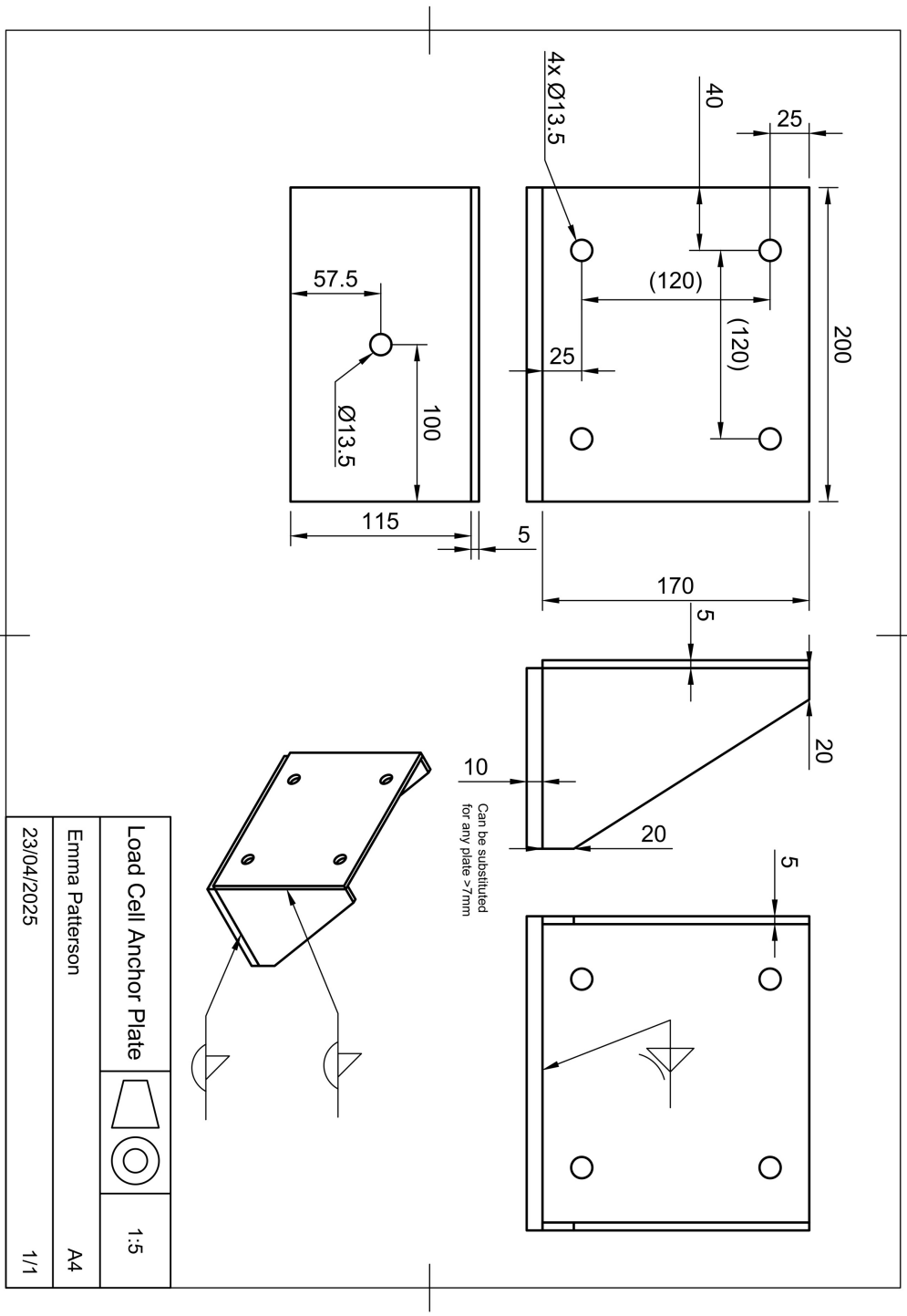

| | | |
|---|---|---|
| FD Support | | |
| Emma Patterson | A4 | 1:5 |
| 07/05/2025 | 1/1 | |

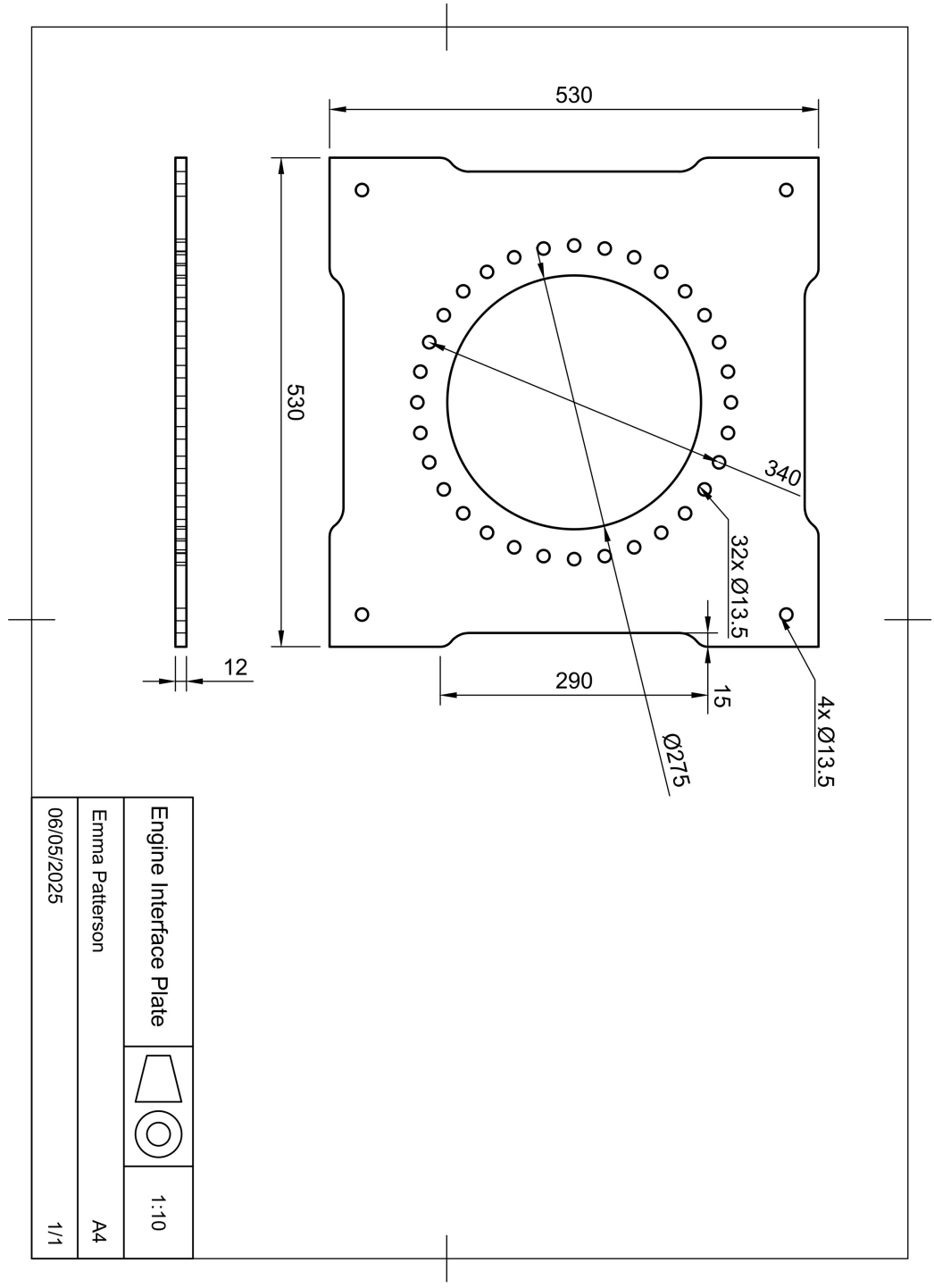

Flame Deflector Scale Model
1:7.5
A4
Emma Patterson
25/04/2025
1/1

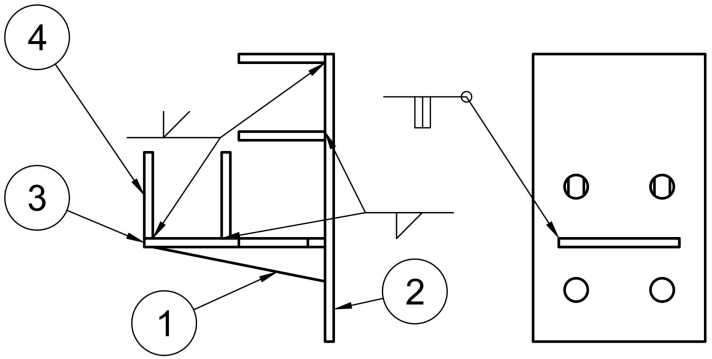
### A.1.2 Thrust Frame



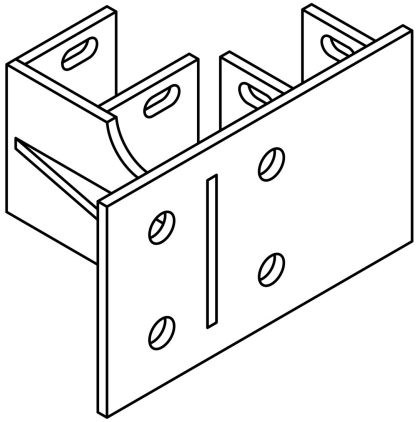

115
100
57.5
Ø13.5
200
5
10
Can be substituted
for any plate >7mm
5
20
60
20
0
Load Cell Bracket
1:5
Emma Patterson
A4
24/04/2025
1/1

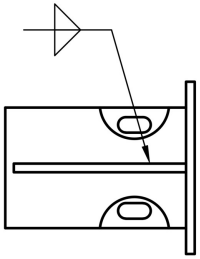

4x Ø13.5
40
25
200
(120)
(120)
25
57.5
Ø13.5
100
115
5
170
5
20
10
20
Can be substituted
for any plate >7mm
5
Load Cell Anchor Plate
1:5
Emma Patterson
A4
23/04/2025
1/1


530
530
340
32x Ø13.5
4x Ø13.5
290
15
Ø275
12
Engine Interface Plate
1:10
Emma Patterson
A4
06/05/2025
1/1





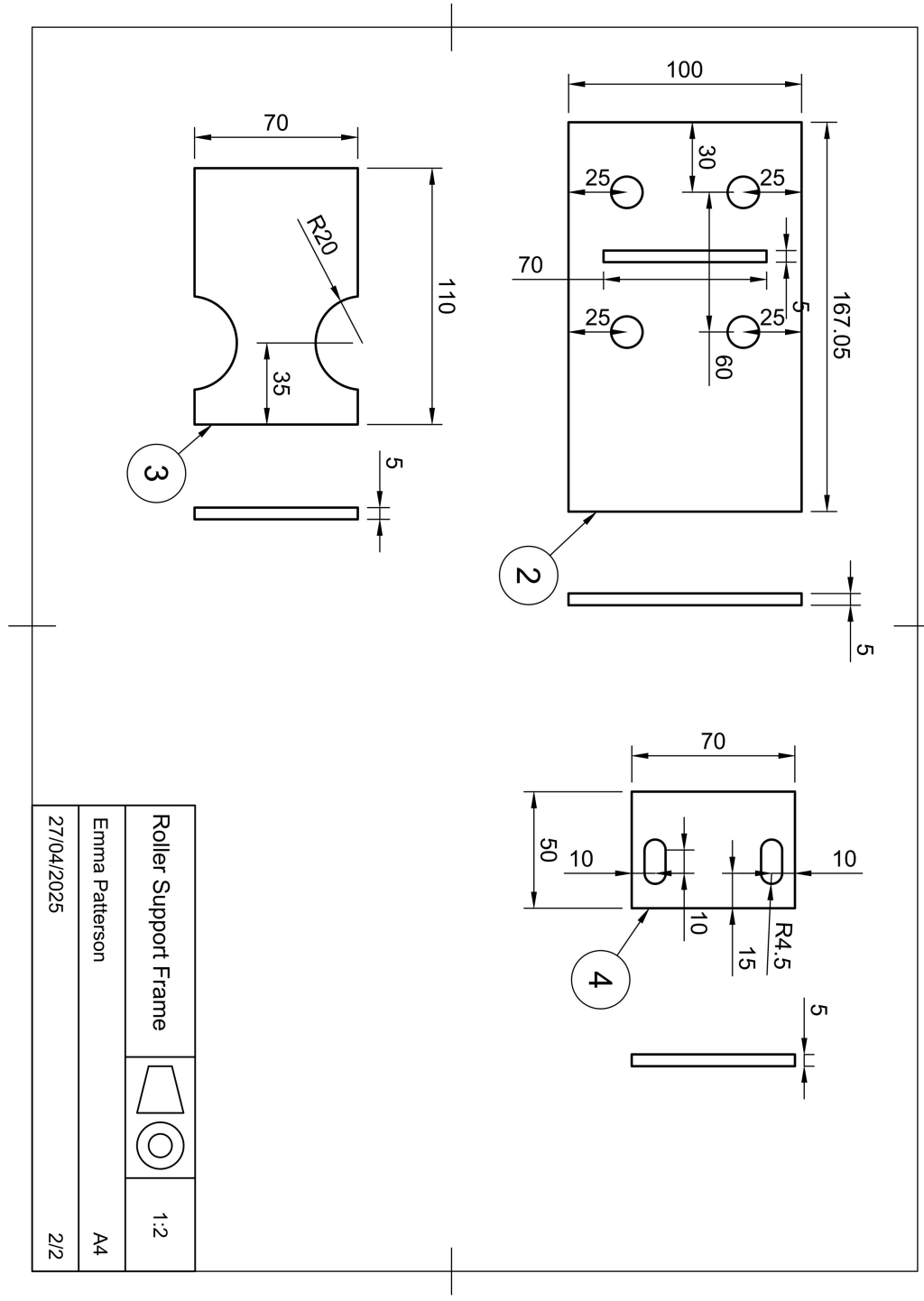



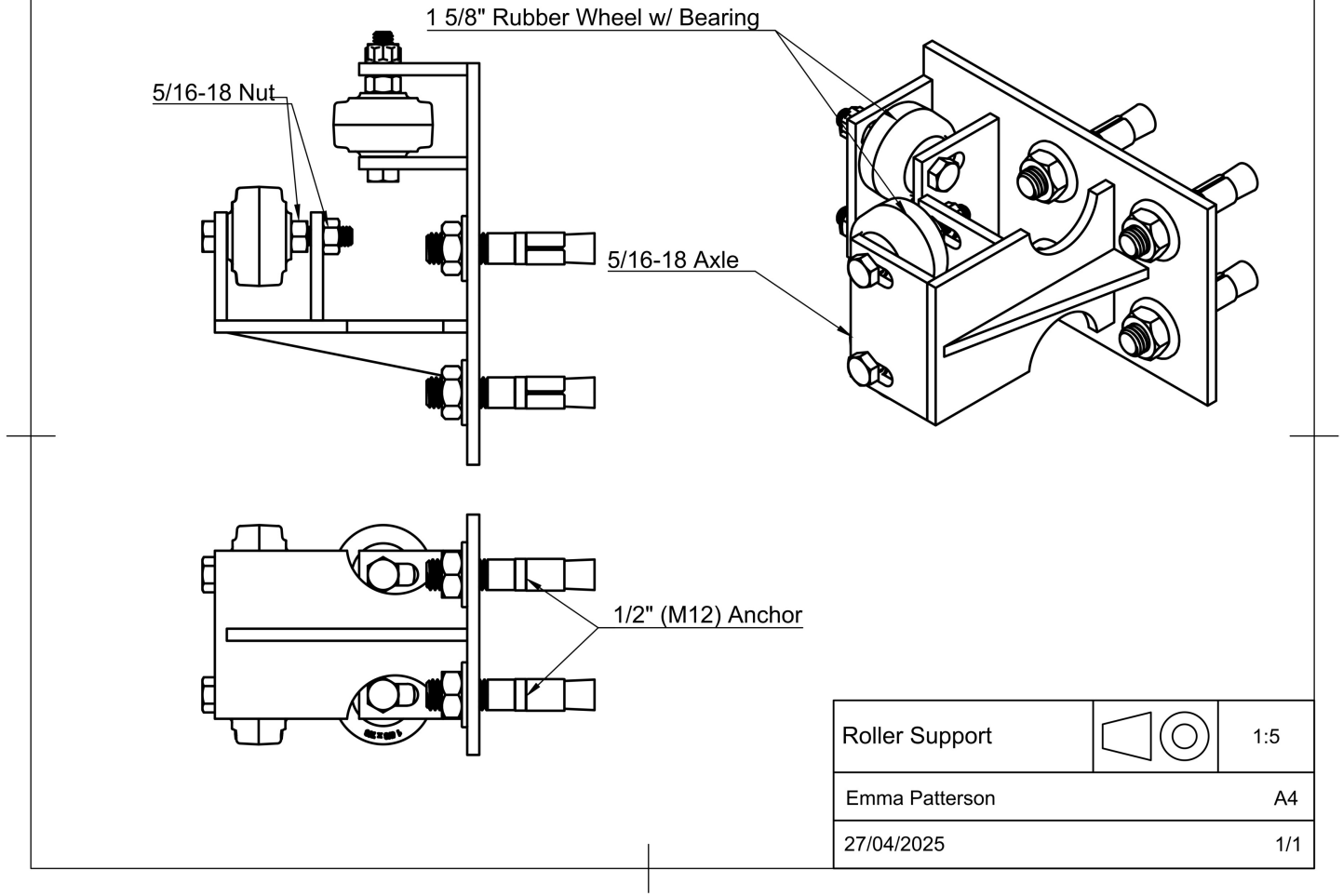


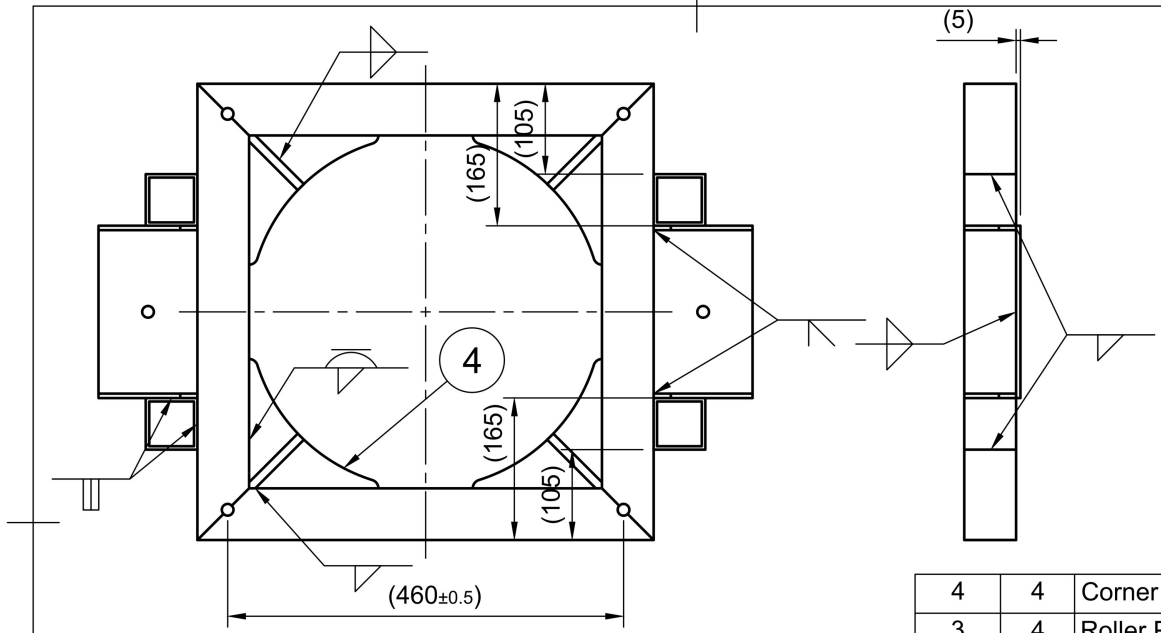
| Item | Qty | Part Number |
|---|---|---|
| 4 | 4 | Interface Plate |
| 3 | 1 | Wheel Mount |
| 2 | 1 | Base Plate |
| 1 | 1 | Brace |

Parts List

| | | |
|---|---|---|
| Roller Support Frame | | 1:2 |
| Emma Patterson | | A4 |
| 27/04/2025 | | 1/2 |


100
30
25
25
70
5
25
25
60
167.05
5
70
R20
110
35
5
3
2
70
50
10
10
10
R4.5
15
4
5
Roller Support Frame
1:2
Emma Patterson
A4
27/04/2025
2/2

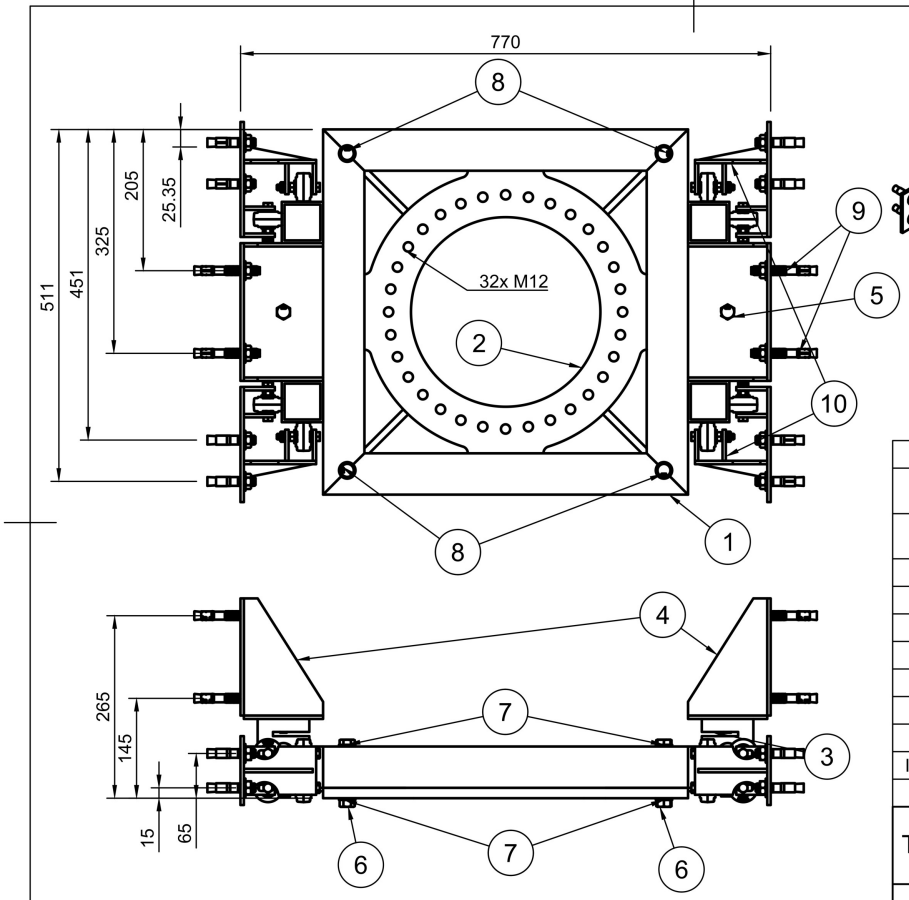

1 5/8" Rubber Wheel w/ Bearing
5/16-18 Nut
5/16-18 Axle
1/2" (M12) Anchor
Roller Support
1:5
Emma Patterson
A4
27/04/2025
1/1

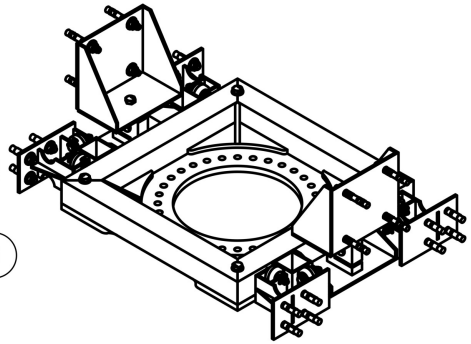



| Item | Qty | Part Number |
|---|---|---|
| 4 | 4 | Corner Support |
| 3 | 4 | Roller Plate |
| 2 | 2 | Load Cell Bracket |
| 1 | 1 | Square Tube Subassembly |

Parts List

| | | |
|---|---|---|
| Thrust Frame Subassembly | | 1:6 |
| Emma Patterson | | A4 |
| 24/04/2025 | | 1/1 |



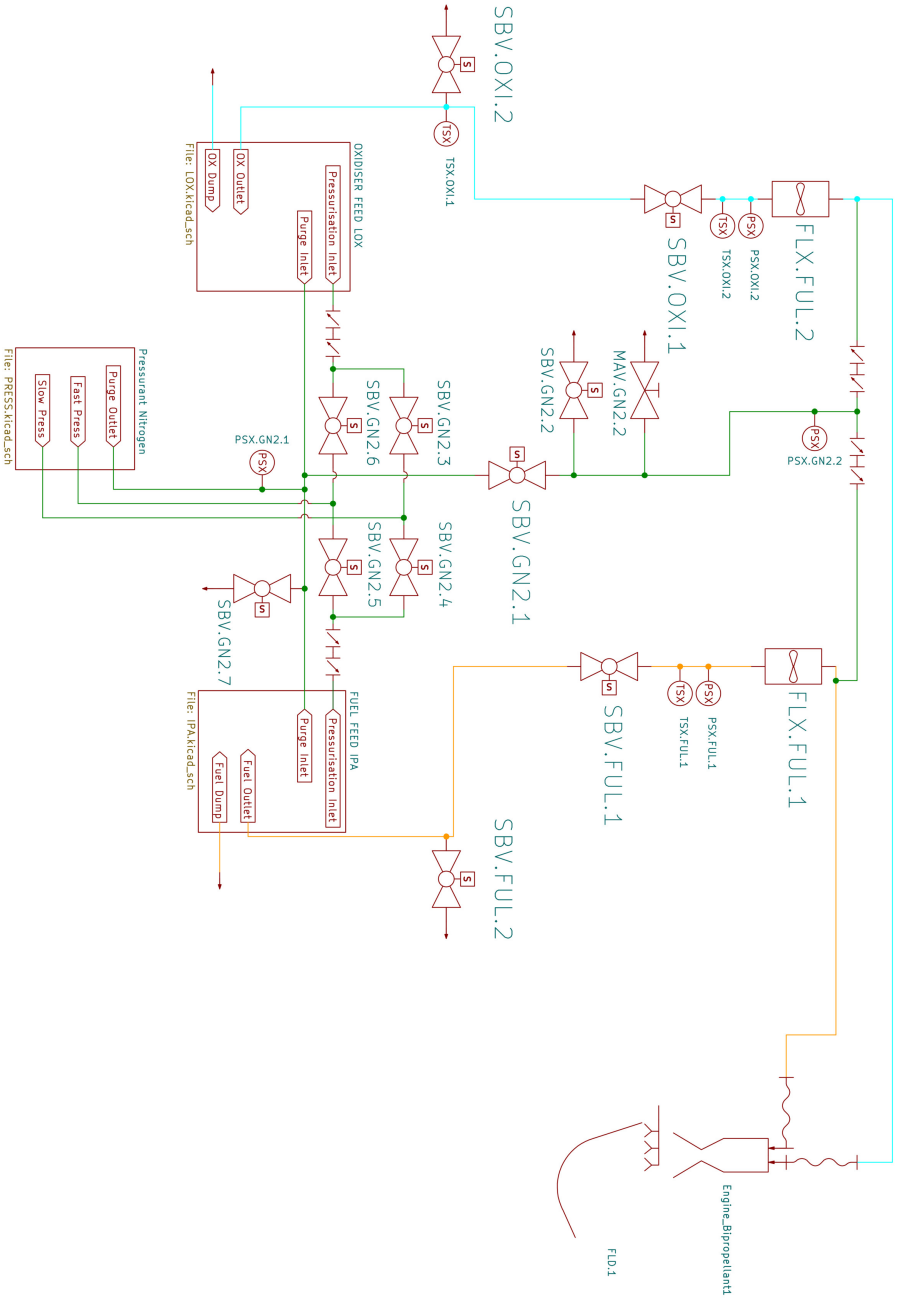
| Item | Qty | Name |
|---|---|---|
| 10 | 4 | Roller Support |
| 9 | 8 | M12 (1/2") 100mm (3 3/4") High Strength Concrete Anchor |
| 8 | 4 | M12X1.75 80mm Partial Thread Grade 10.9 Hex Bolt |
| 7 | 8 | M12 Washer |
| 6 | 4 | M12x1.75 Nut |
| 5 | 4 | M12x1.75 25mm Hex Bolt |
| 4 | 2 | Load Cell Anchor Plate |
| 3 | 2 | S-Type Load Cell |
| 2 | 1 | Engine Interface Plate |
| 1 | 1 | Thrust Frame Subassembly |

Parts List

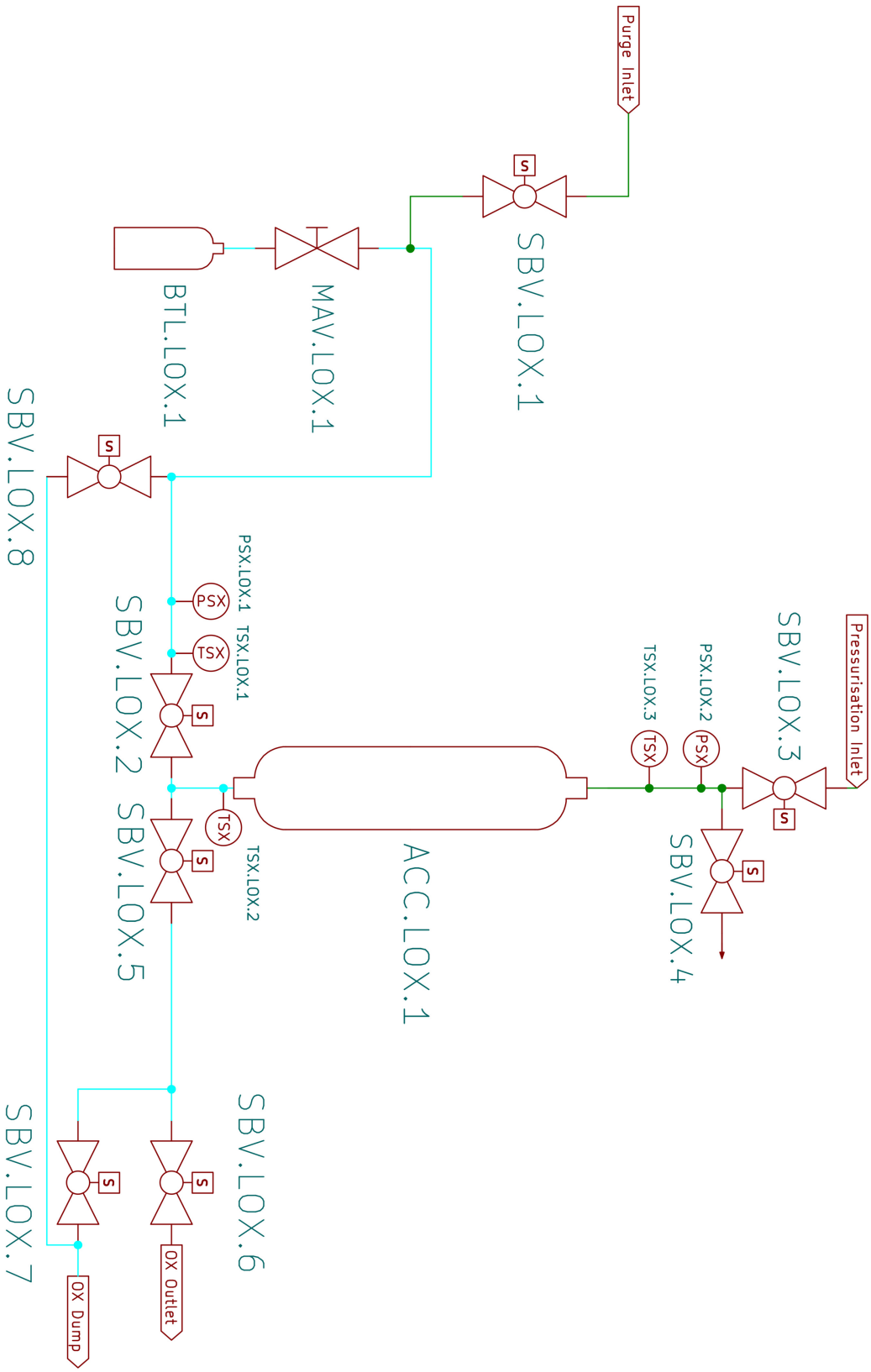
| | | |
|---|---|---|
| Thrust Frame Assembly | | 1:7.5 |
| Emma Patterson | | A4 |
| 24/04/2025 | | 1/1 |

## A.2 P&IDs

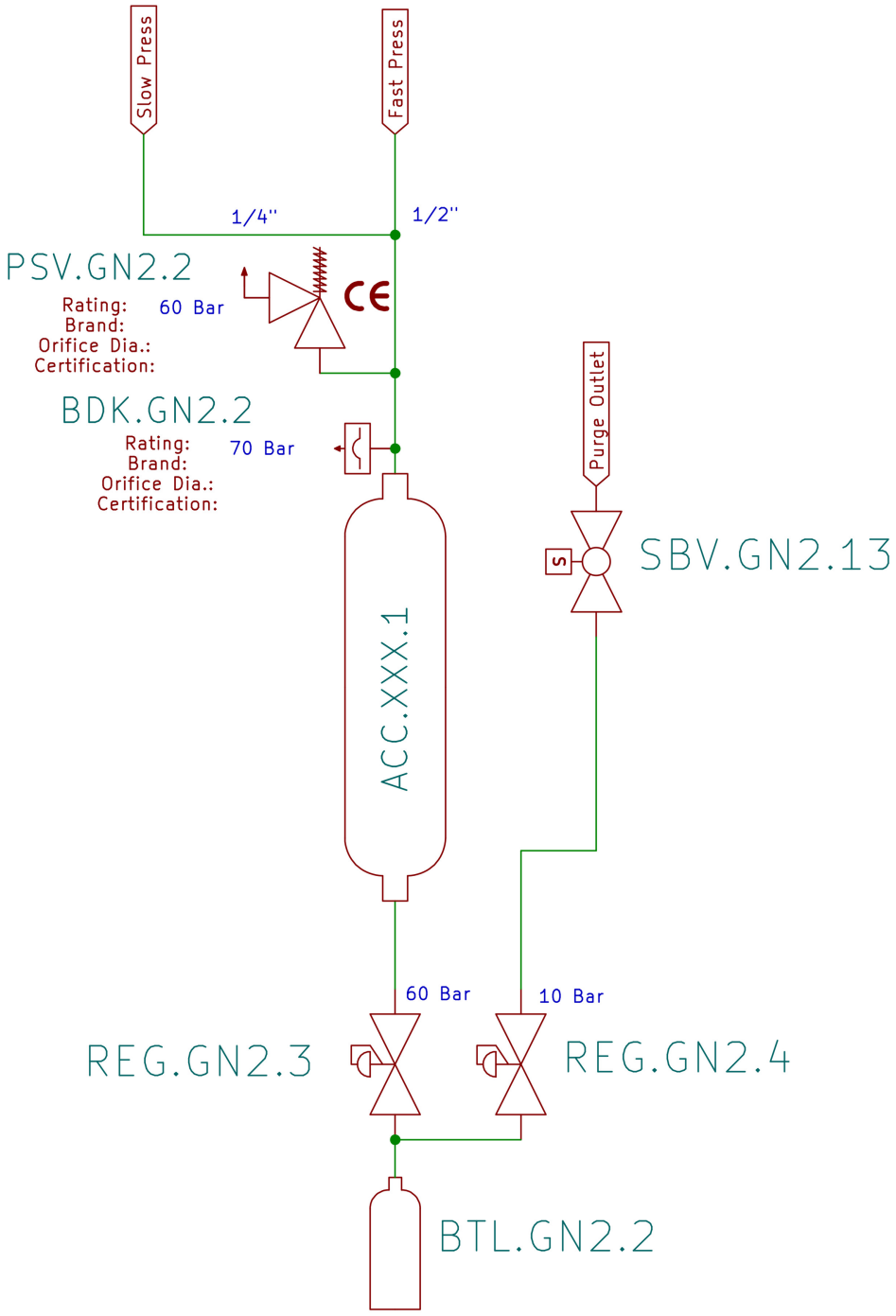
### A.2.1 Top Level



### A.2.2 Liquid Oxygen

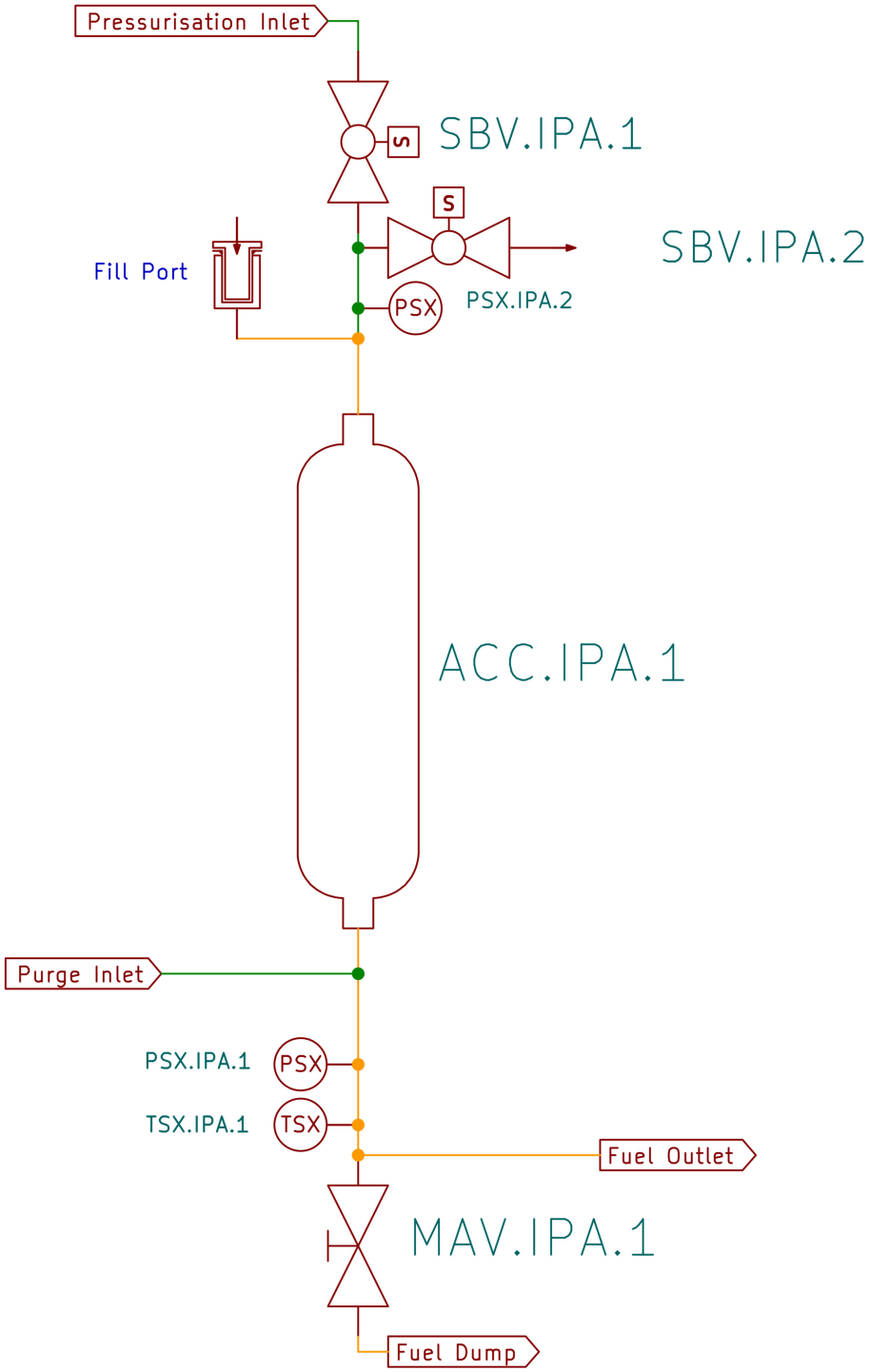


### A.2.3 Pressurisation and Pneumatics

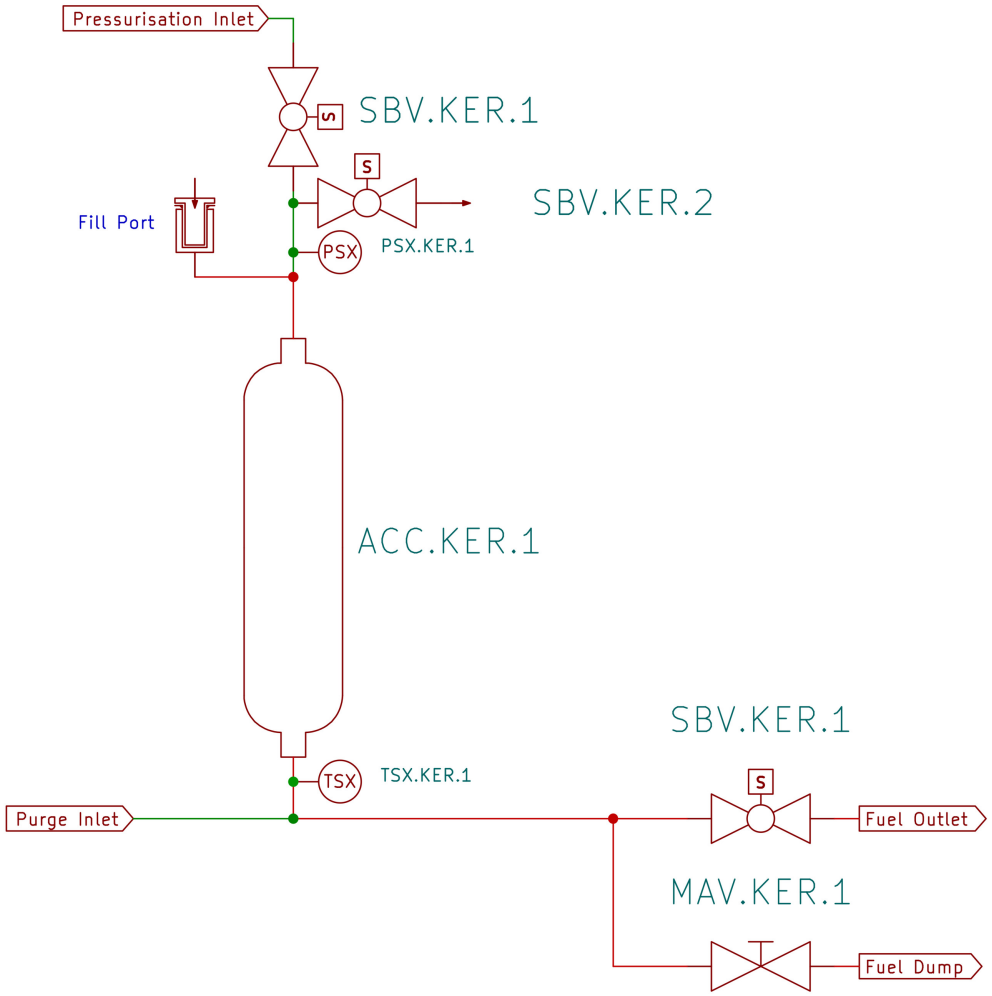


### A.2.4 Fuels

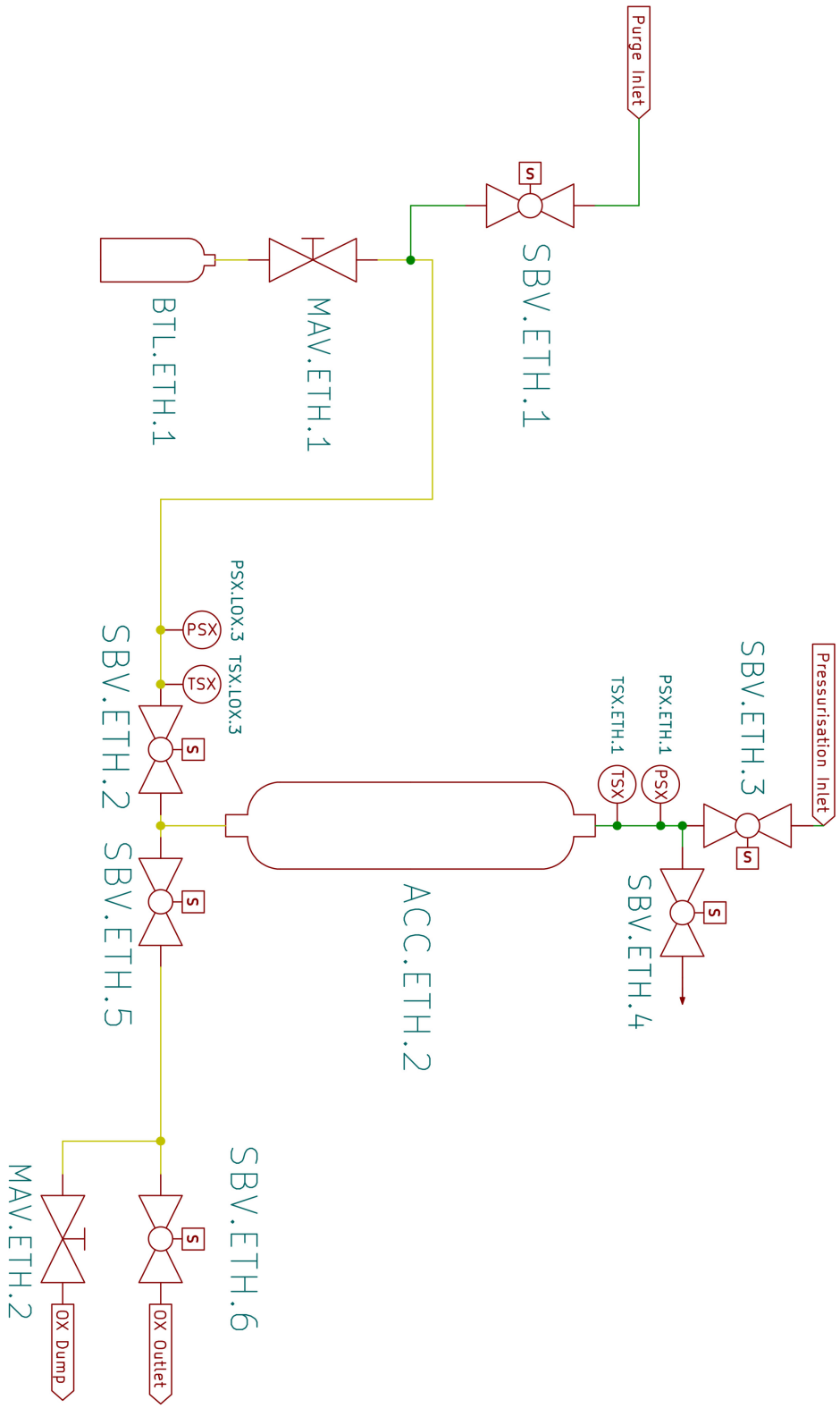
**IPA**



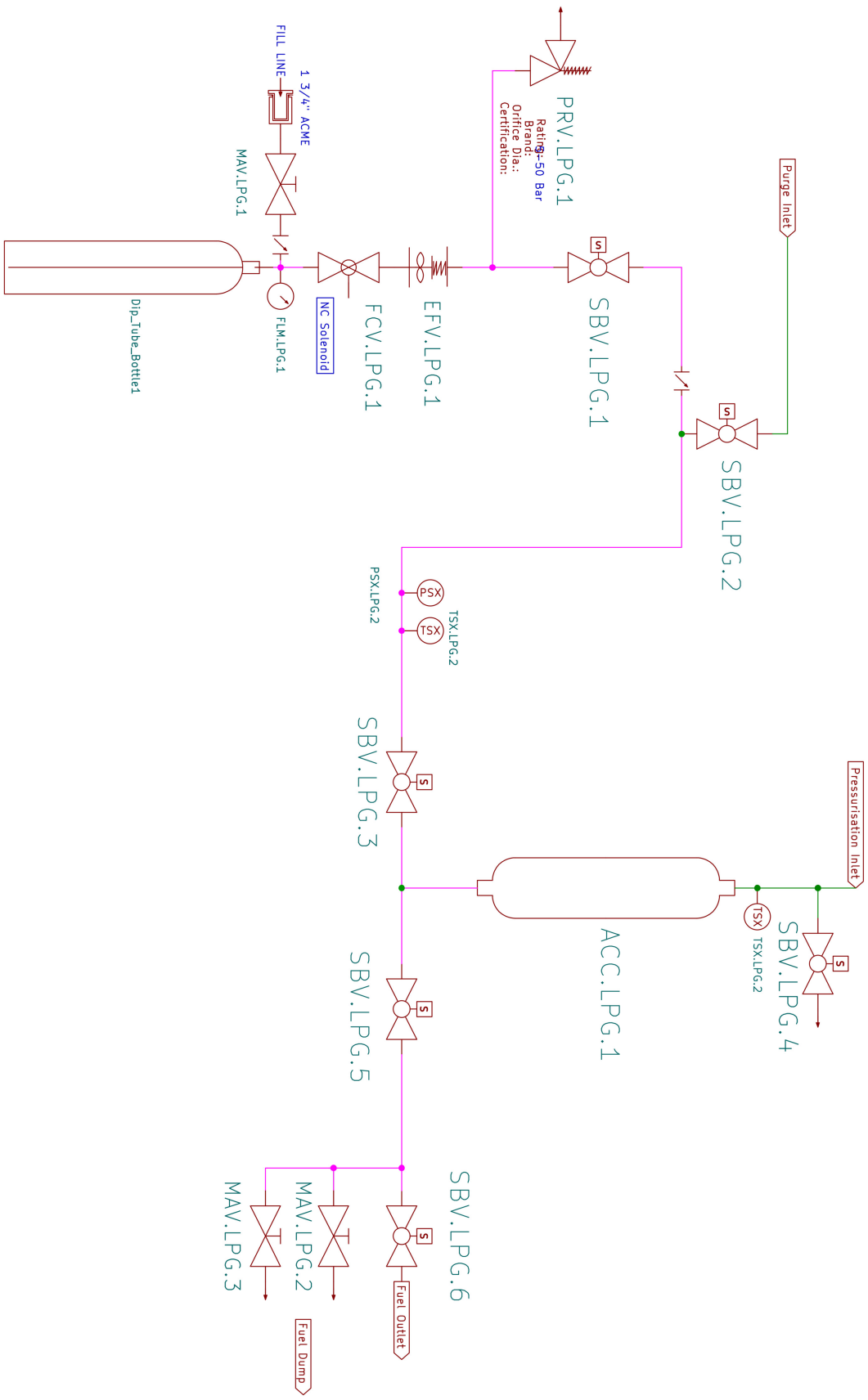
### Kerosene


Pressurisation Inlet
SBV.KER.1
S
S
SBV.KER.2
Fill Port
PSX
PSX.KER.1
ACC.KER.1
SBV.KER.1
TSX
TSX.KER.1
S
Purge Inlet
Fuel Outlet
MAV.KER.1
Fuel Dump

**Ethane**


Purge Inlet
S
SBV.ETH.1
BTL.ETH.1
MAV.ETH.1
PSX.LOX.3
TSX.LOX.3
PSX
TSX
SBV.ETH.2
SBV.ETH.5
ACC.ETH.2
TSX.ETH.1
PSX.ETH.1
TSX
PSX
SBV.ETH.3
Pressurisation Inlet
SBV.ETH.4
SBV.ETH.6
MAV.ETH.2
OX Dump
OX Outlet

## LPG/LNG


PRV.LPG.1
Rating:-50 Bar
Brand:
Orifice Dia.:
Certification:
Purge Inlet
FILL LINE
1 3/4" ACME
MAV.LPG.1
Dip_Tube_Bottle1
FLM.LPG.1
NC Solenoid
FCV.LPG.1
EFV.LPG.1
SBV.LPG.1
SBV.LPG.2
PSX.LPG.2
TSX.LPG.2
SBV.LPG.3
ACC.LPG.1
Pressurisation Inlet
SBV.LPG.4
SBV.LPG.5
SBV.LPG.6
MAV.LPG.3
MAV.LPG.2
Fuel Outlet
Fuel Dump

# Appendix B

# Details of the FLAME Program

## B.1 Class Structures and Functions

### B.1.1 Engine Class

The engine class takes values about the geometry and flow within the rocket engine as an input. These values are expected to be exported from RPA or another similar simulation program. The stand-off distance is also included here as it will vary depending on the test setup and is not a constant value of the deflector. The method `Expand()` uses the provided pressure ratio or exhaust velocity to simulate the flow between the combustion chamber and the exit of the nozzle. This method must be called first as it calculates key flow properties. The `Propagate()` method uses the relationship presented in [2] to evaluate the properties of the plume some distance from the nozzle, and is called during the function of the `Deflect()` method. `Deflect()` is called with the flame deflector object as an input. It uses the impingement angle of the deflector to calculate the oblique shock-wave formed, and therefore the flow conditions after the shock, as these are the conditions that are used for heat flux calculations.

### B.1.2 Deflector Class

The deflector class is used to define key values and properties of the flame deflector. The method `Hotspot()` is included, and uses the temperature-flux relationship given in [1] to quickly estimate the temperature profile of the deflector. A second method of predicting the temperature has been included, `Temp_Predict()` using a data-based approach. FEA data was taken for the temperature at different temperature sensor (TS) locations (TS1, 2, 3, 4) for a given flux. This approach would only be valid for the same geometry as the FEA case, and within the extent of the data produced. As the test cases included up to a 20 second burn and $4MW/m^2$ heat flux. For all reasonable test cases with the scale model deflector, this approach should work well. In order to implement this data-based method on future deflectors, the FEA analysis must be re-done, with the approach taken being to set the internal surface heat flux to each value through a parametric study, and taking the probe point maximum temperature as an output.

### B.1.3 evans_HTC() Function

This function implements the heat transfer coefficient equation from [2]. First, units are converted to imperial, then the equation is applied, and the heat transfer coefficient is converted back to metric to be returned.

### B.1.4 bartz_HFX() Function

This function returns the heat flux estimated by Bartz' method. It is equivalent to the heat flux of a constant-area nozzle extension with the flow conditions that are found after the oblique

shock on the deflector. This is used as a comparison point with Evans' method, and the user is warned if the Bartz result is unused. This occurs when the value of the Bartz result is lower than the minimum value of the Evans result, and can be remedied by increasing the length of the simulation

### B.1.5 eckert_HFX() Function

This function applies the approximations presented by Eckert in [28]. The Reynolds number is calculated, which is then used to estimate the skin friction coefficient, $C_f$. The recovery factor is then determined, and the Stanton number used to find the heat transfer coefficient. The heat flux is calculated using the temperature difference between the wall and the recovery temperature, and the heat transfer coefficient. Two methods to evaluate the skin friction coefficient are provided. Depending on the fluid composition, different methods will agree more closely with Evans. There is an error check for a large deviation between the two temperature methods, and the evaluation method for $C_f$ can be changed to ensure agreement.

### B.1.6 get_strong_weak_oblique() Function

$$tan(\delta) = \frac{2}{\tan\beta}\left(\frac{M_1^2 sin^2\beta - 1}{M_1^2(\gamma + cos(2\beta)) + 2}\right) \tag{B.1}$$

$$cot(\delta)^3 + \left(1\frac{\gamma+1}{2}M_1^2\right)cot(\delta)^2 + (1-M_1^2)cot(\delta) + \left[\left(\frac{\gamma-1}{2}M_1^2 tan(\beta)\right)(2+M_1^2) + (1-M_1^2)\right] = 0 \tag{B.2}$$

This function uses a numerical method to solve the cubic equation B.2 to find values for $cot(\delta)$. The two positive values can then be rearranged, and the maximum and minimum values for $\delta$ returned as the result. The result is the strong and weak shock angle for some impingement into the flow, and can be used to find the properties of the flow before and after the shock.

### B.1.7 Main() Function

The main function contains all of the calculation logic for the order of the functions used. The engine and deflector are initialised, the flow at the nozzle exit calculated and the impingement properties along the deflector calculated. The heat flux is then calculated and depending on user input, the temperature can be calculated and graphed or the direct heat flux graphed. A basic command line interface is included, though the improvement of the usability of the software either by an improvement of the command line interface or the creation of a GUI would increase the usefulness of the software.

## B.2 FLAME source code

### B.2.1 Global functions and Classes

```
from __future__ import annotations
import csv
import datetime
import warnings
import fluids.constants
import matplotlib.pyplot as plt
import numpy as np

start_time = datetime.datetime.now()

class Deflector:
    def __init__(self, impingement_angle, conductivity, density,
```

```
                standoff_distance, cp, diameter, inlet_length, surface_area,
                curvature_radius = None, outlet_length=None):
        self.impingement_angle = impingement_angle
        self.standoff_distance = standoff_distance
        self.conductivity = conductivity
        self.density = density
        self.surface_area = surface_area
        self.cp = cp
        self.dia = diameter
        self.inlet = inlet_length
        if curvature_radius is not None:
            self.curvature = curvature_radius
        else:
            self.curvature = diameter

        if outlet_length is not None:
            self.outlet = outlet_length
        else:
            self.outlet = self.inlet

    def Hotspot(self, heat_flux, time):
        heat_flux_imperial = heat_flux * self.surface_area * 8.81643
        density = self.density * 62.427962
        conductivity = self.conductivity * (0.5777893165/60)
        cp = self.cp * 2.78E-7

        alpha = 2 * heat_flux_imperial * np.sqrt(time)
        beta = np.sqrt(np.pi * density * conductivity * cp)
        peak_temp_R = alpha / beta + 540
        peak_temp_C = peak_temp_R * (5 / 9) - 273.15
        return peak_temp_C

    def Temp_Predict(self, heat_flux, time):
        if not (self.impingement_angle == 10 and
                self.standoff_distance == 0.145 and
                self.dia == 0.127):
            warnings.warn(
                "Data-Based temperature model is only valid "
                "for the test case deflector")
        if len(heat_flux) > 4:
            warnings.warn(
                "Long heat flux value passed to Temp_Predict function, "
                "intended functionality is 4 flux values for the location"
                "of each temp sensor.")
        heat_flux = np.multiply(heat_flux, 1e-6)  # Convert to w/mm2
        # that the data was created from
        if max(heat_flux) > 4 or time > 20:
            warnings.warn(
                "High heat flux or long burn time may make the "
                "data-based prediction inaccurate")
        TC1_Coeffs = [
            time ** 2 * -0.149026848655817 + time * 2.08377049470838 +
                -6.97463670545437,
            time ** 2 * 0.221765953633429 + time * 25.1710233895912 +
                -12.8486632144894,
            time ** 2 * -0.0865827084693638 + time * 1.22941132509 +
                15.8557275263219]
        TC2_Coeffs = [
            time ** 2 * -0.124364079503083 + time * 1.77572220972279 +
                -6.02774567430873,
            time ** 2 * 0.202701169823531 + time * 23.0249103972411 +
                -7.39755011873326,
```

```
70              time ** 2 * -0.0745708016127085 + time * 1.09056122300014 +
                    16.2519419204681]
71          TC3_Coeffs = [
72              time ** 2 * -0.0925031958884924 + time * 1.31003556427163 +
                    -4.43919052835322,
73              time ** 2 * 0.11527909976571 + time * 22.1062051101273 +
                    -7.64403749166789,
74              time ** 2 * -0.0583174162257 + time * 0.858726846458673 +
                    17.0273079699316]
75          TC4_Coeffs = [
76              time ** 2 * -0.0632672296975038 + time * 0.914460478534484 +
                    -3.13346010715919,
77              time ** 2 * 0.0467257436397149 + time * 19.871852759933 +
                    -10.0209987717825,
78              time ** 2 * -0.0411590045829899 + time * 0.625592989733291 +
                    17.7943146211439]
79
80          a1 = [heat_flux[0] ** 2, heat_flux[0], 1]
81          a2 = [heat_flux[1] ** 2, heat_flux[1], 1]
82          a3 = [heat_flux[2] ** 2, heat_flux[2], 1]
83          a4 = [heat_flux[3] ** 2, heat_flux[3], 1]
84
85          probe_temps = [np.dot(a1, TC1_Coeffs), np.dot(a2, TC2_Coeffs),
86                         np.dot(a3, TC3_Coeffs), np.dot(a4, TC4_Coeffs)]
87          return probe_temps
88
89
90  class Engine:
91      def __init__(self, combustor_temp_K, exit_dia, combustor_pressure_pa, gamma,
92                   k, viscosity, cp, massflow,
93                   throat_dia, prandtl, r_comb=None, r_ex=None,
94                   molecular_weight_c=None, molecular_weight_ex=None,
95                   throat_vel=None):
96          self.T_sg_c = combustor_temp_K
97          self.P_sg_c = combustor_pressure_pa
98          self.gamma = gamma
99          self.conductivity = k
100         self.mu = viscosity
101         self.cp = cp
102         self.r_c = r_comb
103         self.r_ex = r_ex
104         self.mw_c = molecular_weight_c
105         self.mw_ex = molecular_weight_ex
106         self.exit_dia = exit_dia
107         self.throat_dia = throat_dia
108         self.throat_vel = throat_vel
109         self.v_ex = None
110         self.a_ex = None
111         self.mdot = massflow
112         self.Pr = prandtl
113
114         if (([self.mw_c, self.mw_ex], [self.r_c, self.r_ex]) ==
115                 ([None,None],[None, None])):
116             raise Exception(
117                 "Definition failure, Please specifiy at least " +
118                 "one molecular weight or value for R\n")
119
120         # account for single mw value
121         if self.mw_ex is None:
122             self.mw_ex = self.mw_c
123         # account for missing r values
124         if self.r_c is None:
125             self.r_c = (fluids.constants.gas_constant * 1000) / self.mw_c
```

```
126
127         if self.r_ex is None:
128             self.r_ex = (fluids.constants.gas_constant * 1000) / self.mw_ex
129
130     def Expand(self, expansion_ratio=None, expansion_pressure=None,
131                exhaust_mach=None):
132         # Nozzle properties
133         nozzle_properties = [expansion_ratio, expansion_pressure, exhaust_mach]
134         for i in range(0, len(nozzle_properties)):
135             if nozzle_properties[i] is not None:
136                 nozzle_properties[i] = 1
137         if nozzle_properties.count(1) > 1:
138             if nozzle_properties[0] == 1:
139                 nozzle_properties[1:] = None
140             elif nozzle_properties[1] == 1:
141                 nozzle_properties[2] = None
142         if nozzle_properties == [0, 0, 0]:
143             if self.throat_dia is not None and self.throat_vel is not None:
144                 area_ratio = self.exit_dia / self.throat_dia
145                 Ma_ex = np.sqrt((2 / (self.gamma - 1)) * ((area_ratio ** (
146                         (self.gamma - 1) / self.gamma)) - 1))
147                 stag_pressure_ratio = (1 + ((self.gamma - 1) / 2) * (
148                         Ma_ex ** 2)) ** (self.gamma / (self.gamma - 1))
149                 self.Psa_ex = self.P_sg_c / stag_pressure_ratio
150                 self.Tsa_e = self.T_sg_c / (
151                         1 + (((self.gamma - 1) / 2) * Ma_ex ** 2))
152                 self.a_ex = np.sqrt(self.r_ex * self.Tsa_e)
153                 self.v_ex = self.a_ex * Ma_ex
154             else:
155                 raise Exception(
156                     "Specify nozzle properties when calling the Expand function"
157                     + ", or exit ratio during engine definition\n")
158         match nozzle_properties:
159             case [1, None, None]:
160                 self.Psa_ex = expansion_ratio * 101325
161                 Ma_ex = np.sqrt((((self.P_sg_c / self.Psa_ex) ** (
162                         (self.gamma - 1) / self.gamma)) - 1) * 2 / (
163                                         self.gamma - 1))
164                 self.Tsa_e = self.T_sg_c / (
165                         1 + (((self.gamma - 1) / 2) * Ma_ex ** 2))
166                 self.a_ex = np.sqrt(self.r_ex * self.Tsa_e)
167                 self.v_ex = self.a_ex * Ma_ex
168             case [None, 1, None]:
169                 self.Psa_ex = expansion_pressure
170                 Ma_ex = np.sqrt((((self.P_sg_c / self.Psa_ex) ** (
171                         (self.gamma - 1) / self.gamma)) - 1) * 2 / (
172                                         self.gamma - 1))
173                 self.Tsa_e = self.T_sg_c / (
174                         1 + (((self.gamma - 1) / 2) * Ma_ex ** 2))
175                 self.a_ex = np.sqrt(self.r_ex * self.Tsa_e)
176                 self.v_ex = self.a_ex * Ma_ex
177             case [None, None, 1]:
178                 Ma_ex = exhaust_mach
179                 stag_pressure_ratio = (1 + ((self.gamma - 1) / 2) * (
180                         Ma_ex ** 2)) ** (self.gamma / (self.gamma - 1))
181                 self.Psa_ex = self.P_sg_c / stag_pressure_ratio
182                 self.Tsa_e = self.T_sg_c / (
183                         1 + (((self.gamma - 1) / 2) * Ma_ex ** 2))
184                 self.a_ex = np.sqrt(self.r_ex * self.Tsa_e)
185                 self.v_ex = self.a_ex * Ma_ex
186             case _:
187                 raise Exception("Please specify exhaust properties\n")
188
```

```
189     def Propagate(self, distance, plume_dia=None):
190         if self.v_ex is None:
191             raise Exception(
192                 "Please use the Expand function before calling Propagate\n")
193         if distance < ((self.exit_dia / 2) / 0.053):
194             return self.v_ex
195         if plume_dia is None:
196             plume_dia = self.exit_dia
197         v_x = ((10 ** 0.79 - (33 * ((plume_dia ** 2) / distance ** 2)))
198                / (distance / plume_dia)) * self.v_ex
199         return v_x
200 
201     def Deflect(self, D: Deflector):
202         standoff_distance = D.standoff_distance
203         impingement_angle = D.impingement_angle
204 
205         v_impingement = self.Propagate(
206             standoff_distance)  # impingement after standoff angle,
207                                 # at centre of plume
208         self.v_impingement = v_impingement
209 
210         a_ex = np.sqrt(self.r_ex * self.Tsa_e)
211         mach_at_impingement = v_impingement / a_ex  # velocity change will be
212         # small, so anticipated
213         # speed of sound change is low
214 
215         sigma = get_strong_weak_oblique(mach_at_impingement, impingement_angle,
216                                         self.gamma)[0]
217         # gets the weak shock angle for the impingement that's provided by the FD
218         freestream_shock_angle = 180 - (
219                 90 + impingement_angle + sigma)
220         # Angle from freestream to shock normal in deg.
221         mach_normal_1 = mach_at_impingement * np.cos(np.deg2rad(
222             freestream_shock_angle))
223         # take the mach speed normal to the shock angle
224         mach_tangent = mach_at_impingement * np.sin(
225             np.deg2rad(freestream_shock_angle))
226         mach_normal_2 = np.sqrt(
227             (1 + ((self.gamma - 1) / 2) * mach_normal_1 ** 2) /
228             ((self.gamma * mach_normal_1 ** 2) - ((self.gamma - 1) / 2))
229         )  # find mach normal to shock angle after oblique shock
230 
231         mach_after_impingement = np.sqrt(mach_normal_2 ** 2 + mach_tangent ** 2)
232 
233         Tsa_after_oblique = self.Tsa_e * (
234                 (1 + ((2 * self.gamma) / (self.gamma + 1)) * (
235                         mach_normal_1 ** 2 - 1)) *
236                 ((2 + (self.gamma - 1) * mach_normal_1 ** 2) / (
237                         (self.gamma + 1) * mach_normal_1 ** 2))
238         )
239 
240         speed_of_sound_after_impingement = np.sqrt(
241             self.r_ex * self.gamma * Tsa_after_oblique)
242         # speed of sound after temp change through shock
243 
244         speed_along_FD = mach_after_impingement * speed_of_sound_after_impingement
245         self.V_post_shock = speed_along_FD
246         self.T_st_post_shock = Tsa_after_oblique
247 
248         return
249 
250 
251 def Temp_Surface_Conversion(D: Deflector, time, probe_Temperatures):
```

```
252     if not (D.impingement_angle == 10 and
253             D.standoff_distance == 0.145 and
254             D.dia == 0.127):
255         warnings.warn(
256             "Data-Based temperature model is only valid"
257             " for the test case deflector\n")
258     TC1_Coeffs = [
259         time ** 2 * 7.09261637335221E-15 + time * -1.38876724441972E-12 +
                -3.09030135958868E-12,
260         time ** 2 * 0.00472560170362053 + time * -0.154571858432344 +
                2.41640356729064,
261         time ** 2 * -0.0945120080353407 + time * 3.09143654466126 +
                -28.3280696393145]
262     TC2_Coeffs = [
263         time ** 2 * -2.10274038265068E-12 + time * 5.09804975848031E-11 +
                -2.16765908380359E-10,
264         time ** 2 * 0.00264595345324568 + time * -0.0843497980285974 +
                1.66532103113641,
265         time ** 2 * -0.0529193963832935 + time * 1.68700373603717 +
                -13.3064499357725]
266     TC3_Coeffs = [
267         time ** 2 * -1.11933104904635E-12 + time * 2.85379914652035E-11 +
                -1.60409044742089E-10,
268         time ** 2 * 0.00261729031216522 + time * -0.0825824117622711 +
                1.62532365923783,
269         time ** 2 * -0.0523460955072181 + time * 1.65165521633515 +
                -12.5065083783402]
270     TC4_Coeffs = [
271         time ** 2 * 7.00963467615162E-13 + time * -2.34379496561615E-11 +
                1.94274528890156E-10,
272         time ** 2 * 0.00249198956837728 + time * -0.0782763761646628 +
                1.58512963481065,
273         time ** 2 * -0.0498398230613657 + time * 1.56552772525774 +
                -11.7025881417731]
274
275     a1 = [probe_Temperatures[0] ** 2, probe_Temperatures[0], 1]
276     a2 = [probe_Temperatures[1] ** 2, probe_Temperatures[1], 1]
277     a3 = [probe_Temperatures[2] ** 2, probe_Temperatures[2], 1]
278     a4 = [probe_Temperatures[3] ** 2, probe_Temperatures[3], 1]
279
280     surface_Temperatures = [np.dot(a1, TC1_Coeffs), np.dot(a2, TC2_Coeffs),
281                             np.dot(a3, TC3_Coeffs), np.dot(a4, TC4_Coeffs)]
282     return surface_Temperatures
283
284
285 def evans_HTC(E: Engine, distance_from_impingement_m):
286     # convert all units to us customary
287     cp = E.cp * 1000 * 2.402
288     # cp BTU/ lb °F
289     conductivity = E.conductivity * 0.5781759824
290     # conductivity Btu (th) foot/ hour ft^2 °F
291     density = (E.Psa_ex * 0.062427962) / (E.r_ex * E.T_sg_c)
292     # density lb/ft3
293     viscosity = E.mu * 17.8579688
294     # dynamic viscosity lb/ in s
295     velocity = E.V_post_shock * 3.280839895
296     # velocity ft/s
297     distance_from_impingement = distance_from_impingement_m * 3.280839895
298     # distance ft
299
300     heat_transfer_coefficient_imperial = ((0.0296 *
301         (((density * velocity * distance_from_impingement) / viscosity) ** (
302                             4 / 5))* (((cp * viscosity) / conductivity) ** (
```

```
                            1 / 3)) * conductivity)/ distance_from_impingement)
    # heat transfer coefficient btu / in^2 sec °F

    heat_transfer_coefficient_metric = (
            heat_transfer_coefficient_imperial * 8.611128)
    return heat_transfer_coefficient_metric


def bartz_HFX(T_Wall, E: Engine):
    cp = E.cp * 1000 * 0.2390057361
    # cp BTU / lb °F
    viscosity = E.mu * 0.000145
    # viscocity lbf sec / in
    throat_Dia = E.throat_dia * 39.3701
    # throat diameter in
    Pr = E.Pr
    T_ex = (E.Tsa_e) * (9 / 5)
    # exhaust temperature °R
    T_Wall = (T_Wall) * (9 / 5)
    # wall temperature °R
    Ma = E.v_ex / E.a_ex
    Cstar = ((E.P_sg_c * (
            np.pi * (E.throat_dia / 2) ** 2)) / E.mdot) * 3.280839895
    # C* ft/s
    g = 32.2
    # gravitation acceleration ft/s^2
    PC = E.P_sg_c * 0.000145038
    # Chamber Pressure lbf / in^2
    gamma = E.gamma
    exit_Dia = E.exit_dia * 39.3701
    # Exit Diameter in

    sigma = ((((T_Wall / (2 * T_ex)) * (
            1 + ((gamma - 1) / 2 * Ma ** 2))) + 0.5) ** (-0.68) * (
                    (1 + ((gamma - 1) / 2 * Ma ** 2)) ** (-0.12)))

    T_effective = T_ex * ((1 + ((Pr ** 0.33) * ((gamma - 1) / 2 * Ma ** 2))) / (
            1 + ((gamma - 1) / 2 * Ma ** 2)))

    heat_transfer_coefficient = ((((0.026) / (throat_Dia ** 0.2)) * (
            (viscosity ** 0.2 * cp) / (Pr ** 0.6)) *
                          ((PC * g) / Cstar) ** 0.8) *
                         ((((throat_Dia / 2) ** 2) * np.pi) /
                          (((exit_Dia / 2) ** 2) * np.pi) ** 0.9) * sigma)
    # heat transfer coefficient BTU/ in sec °F

    factor = (11356.526682 / 12)

    heat_flux_metric = factor * (heat_transfer_coefficient) * (
        (T_effective - T_Wall))

    return heat_flux_metric


def eckert_HFX(T_wall, E: Engine):
    Pr = E.Pr
    V = E.V_post_shock
    Cp = E.cp * 1000
    rho = (E.Psa_ex) / (E.r_ex * E.T_sg_c)
    mu = E.mu
    Re = (rho * V) / mu
    T_s = E.T_st_post_shock
    T_tot = E.T_sg_c
```

```
366
367     Cf = 0.0296 / (Re ** (0.2))
368     # lower values, Blasius equation
369     # Cf = 0.37/((np.log10(Re))**2.53)
370     # higher values of heat flux, schlutz-grunow
371
372     r = Pr ** (1 / 3)
373     T_r = (r * (T_tot - T_s)) + T_s
374     St = (Cf / 2) * (Pr ** -(2 / 3))
375
376     h = St * (rho * Cp * V)
377     flux = h * (T_r - T_wall)
378     return flux
379
380
381 def get_strong_weak_oblique(mach1, delta, gamma=1.4):
382     delta = np.deg2rad(delta)
383     # calculate coefficients
384     A = mach1 ** 2 - 1
385     B = 0.5 * (gamma + 1) * mach1 ** 4 * np.tan(delta)
386     C = (1 + 0.5 * (gamma + 1) * mach1 ** 2) * np.tan(delta)
387     coeffs = [1, C, -A, (B - A * C)]
388
389     # roots of a cubic equation, two positive solutions
390     roots = np.array([r for r in np.roots(coeffs) if r > 0])
391
392     thetas = np.arctan(1 / roots)
393     theta_weak = np.min(thetas)
394     theta_strong = np.max(thetas)
395     return [np.rad2deg(theta_weak), np.rad2deg(theta_strong)]
```

### B.2.2 Scale Model Deflector test case

```
1     def Main():
2     T_initial = 20 + 273.15  # Initial deflector temperature,K
3     total_Length = 800 / 1000  # total length to determine flux, m
4     evans_Startpoint = 2 / 1000  # start of evans approximation, m
5     step_Distance = 1 / 1000  # step setting, m
6
7     array_Length = int((total_Length) / step_Distance)
8     evans_Length = int((total_Length - evans_Startpoint) / step_Distance)
9
10    Baldrick = Engine(combustor_temp_K=1768.7,
11                      exit_dia=48.77 / 1000,
12                      combustor_pressure_pa=2000000,
13                      gamma=1.279,
14                      k=0.1307,
15                      cp=4.4775,
16                      viscosity=0.00004084,
17                      molecular_weight_c=19.4848,
18                      molecular_weight_ex=19.8018,
19                      massflow=0.75,
20                      throat_dia=24.34 / 1000,
21                      prandtl=0.5451)
22    # Values for Baldrick
23
24    Rosie = Engine(combustor_temp_K=2066.97,
25                   exit_dia=39.33 / 1000,
26                   combustor_pressure_pa=2000000,
27                   gamma=1.1279,
```

```
                    k=0.2271,
                    cp=1.92,
                    viscosity=0.0000682,
                    molecular_weight_c=20.1957,
                    molecular_weight_ex=20.2180,
                    massflow=0.55301,
                    throat_dia=22.02 / 1000,
                    prandtl=0.5542)

    Test_Engine = Baldrick  # Choose which engine to use

    Scale_Model = Deflector(impingement_angle=10,
                            conductivity=45,
                            density=7850,
                            standoff_distance=145 / 1000,
                            cp=540,
                            diameter=127 / 1000,
                            inlet_length=(100 - 57)/1000,
                            surface_area=0.0722 + (2*0.0399),
                            curvature_radius=63.5 /1000,
                            outlet_length = 100/1000)
    # Values for scale model deflector currently at Buxton

    Test_Deflector = Scale_Model

    Test_Engine.Expand(expansion_ratio=1)
    Test_Engine.Deflect(Test_Deflector)

    distance_spread = np.linspace(evans_Startpoint, total_Length,
                                  evans_Length)

    evans_Hs = np.zeros(np.shape(distance_spread))

    for i in range(0, len(distance_spread)):
        evans_Hs[i] = evans_HTC(Test_Engine, distance_spread[i])

    bartz_Flux = bartz_HFX(T_initial, Test_Engine)
    eckert_Flux = eckert_HFX(T_initial, Test_Engine)

    evans_Fluxes = np.multiply(evans_Hs, np.sqrt(
        (T_initial - Test_Engine.T_st_post_shock) ** 2))

    valid_Output = False
    units = ""
    while not (valid_Output):
        output_Type = input(
            "Enter type of results to output, (F)lux or "
            "(T)emperature, or (O)ther?\n").lower()
        if output_Type == "flux" or output_Type == "f":
            valid_Output = True
            units = "W"

            evans_Results = evans_Fluxes
            bartz_Result = bartz_Flux
            eckert_Result = eckert_Flux
            Sensor_Temps = None
        elif (output_Type == "other" or
              output_Type == "o"):
            while True:
                input_other = input("Please Enter a command: \n")
                match input_other.lower():
                    case "help" | "h":
                        print("Commands are: (Sensor): calculate surface "
```

```
                            "temp from sensor values\n")
                        # include other commands and functionallity here
                    case "sens" | "Sensor":
                        [time, temp1, temp2, temp3, temp4] = (
                            input("Please input the burn time and sensor temps "
                                  "in the format x,y,z,a,b\n").split())

                        Surface_Temps = Temp_Surface_Conversion(Test_Deflector,
                            time,[temp1, temp2,temp3, temp4])

                        print("Surface temperature is " + str(
                            Surface_Temps) + "°C\n")
                        exit()
                    case _:
                        print("Please enter a command, or '(h)elp' for a "
                              "list of commands\n")
        elif (output_Type == "temperature" or
              output_Type == "t" or output_Type == "temp\n"):
            burn_Valid = False
            while not (burn_Valid):
                burn_Input = input(
                    "Please enter a burn time in seconds: \n")
                try:
                    burn_Time = float(burn_Input)
                    burn_Valid = True
                except:
                    warnings.warn("Please enter an number for burn time\n")
                    burn_Valid = False
            type_Chosen = False
            while not (type_Chosen):
                temp_Type = input(
                    "Please enter the model to use, "
                    "(D)ata based or (P)hysics based: \n").lower()
                if (temp_Type == "physics" or
                        temp_Type == "physics based" or temp_Type == "p"):
                    valid_Output = True
                    type_Chosen = True
                    units = "°C"

                    Sensor_Temps = None
                    evans_Results = Test_Deflector.Hotspot(evans_Fluxes,
                                                           burn_Time)
                    bartz_Result = Test_Deflector.Hotspot(bartz_Flux, burn_Time)
                    eckert_Result = Test_Deflector.Hotspot(eckert_Flux,
                                                           burn_Time)

                elif (temp_Type == "data" or
                      temp_Type == "data based" or temp_Type == "d"):
                    valid_Output = True
                    type_Chosen = True
                    units = "°C"

                    Sensor_Temps = [0, 0, 0, 0]  # as sensor_temps is not None,
                    # we can assign the fluxes later on
                    evans_Results = evans_Fluxes
                    bartz_Result = bartz_Flux
                    eckert_Result = eckert_Flux
                else:
                    print("Invalid command.\n")
        else:
            print("Invalid command.\n")

    # match the different flux or temp values together,
```

```
    #                   while spotting some common issues
    deflector_Results = np.ones(array_Length)

    evans_Start_Index = int(evans_Startpoint / step_Distance)
    deflector_Results[evans_Start_Index:(evans_Start_Index + evans_Length)] = (
            deflector_Results[evans_Start_Index:(
                    evans_Start_Index + evans_Length)] * evans_Results)

    deflector_Results[0:evans_Start_Index + len(
        evans_Results[evans_Results > eckert_Result])] = eckert_Result

    deflector_Results[(evans_Start_Index + evans_Length) - len(
        evans_Results[evans_Results < bartz_Result]):] = bartz_Result
    if deflector_Results[array_Length - 1] != (
            bartz_Result):
        warnings.warn(
            "Bartz result unused, length of flux "
            " evaluation should be increased",
            category=UserWarning)
    if (np.average(evans_Results[0:20]) - eckert_Result) > (eckert_Result / 5):
        warnings.warn(
            "Large disagreement between Evans and Eckert detected, "
            "consider changing the evaluation method of the skin friction "
            "coefficient within the eckert_HFX function",
            category=UserWarning)

    if Sensor_Temps is not None:
        Sensor_Temps = Test_Deflector.Temp_Predict([deflector_Results[0],
                                    deflector_Results[int((
                                              40 / 1000) / step_Distance)],
                                    deflector_Results[int((
                                              100 / 1000) / step_Distance)],
                                    deflector_Results[int((
                                              250 / 1000) / step_Distance)]],
                                   burn_Time)

        Surface_Temps = Temp_Surface_Conversion(Test_Deflector, burn_Time,
                                                Sensor_Temps)

        # Find the sensor temps from the flux at each location.
        # The results arrays are always flux if data-based has been selected

    distance_list = np.linspace(0, total_Length * 1000, array_Length)
    draw_Graph = input(
        "Draw data graph y/n? \n").lower()
    if draw_Graph == "y":
        if Sensor_Temps is not None:
            fig, ax = plt.subplots()
            deflector_Results = deflector_Results / 1000
            ax.plot([0, 40, 100, 250], Sensor_Temps, "o")

            plt.xlabel("Distance from impingement point, mm")
            plt.ylabel("Temperature Probe Temperatures, °C")
            plt.show()
        else:
            fig, ax = plt.subplots()
            deflector_Results = deflector_Results / 1000
            units = "kW"
            ax.plot(distance_list, deflector_Results)

            plt.xlabel("Distance from impingement point, mm")

            if output_Type == "flux":
```

```
                plt.ylabel("Heat flux on deflector, " + units)
            else:
                plt.ylabel("Temperature along deflector, " + units)

            plt.show()

    write_CSV = input(
        "Export data as .csv y/n? \n").lower()
    if write_CSV == "y":
        if Sensor_Temps != None:
            print("Sensor Temperatures are: " + str(Sensor_Temps) + "°C\n")
            print("Surface Temperatures are: " + str(Surface_Temps) + "°C\n")
            print(
                "Done! in " + str(datetime.datetime.now() - start_time) + "s\n")
            return
        Fields = ["Distance (mm)", "Heat Flux (" + units + ")"]
        Data = np.column_stack((distance_list, deflector_Results))
        current_time = str(datetime.datetime.now())[:16].replace(
            " ", "-").replace(":", "-")
        with open(str(output_Type) + "_" + current_time +
                  "_TestCase_.csv", "w", newline="") as f:
            csv_writer = csv.writer(f, dialect="excel")
            csv_writer.writerow(Fields)
            for i in range(0, np.shape(Data)[0]):
                csv_writer.writerow(Data[i, :])

    print("Done! in " + str(datetime.datetime.now() - start_time) + "s\n")
    exit()


if __name__ == '__main__':
    Main()
```

### B.2.3 Theoretical Engine Case

```
def Main():
    T_initial = 20 + 273.15  # Initial deflector temperature,K
    total_Length = 250 / 1000  # total length to determine flux, m
    evans_Startpoint = 2 / 1000  # start of evans approximation, m
    step_Distance = 1 / 1000  # step setting, m

    array_Length = int((total_Length) / step_Distance)
    evans_Length = int((total_Length - evans_Startpoint) / step_Distance)

    IPA_N2O = Engine(combustor_temp_K=3204.0548,
                     exit_dia=86.88 / 1000,
                     combustor_pressure_pa=3000000,
                     gamma=1.2148,
                     k=0.1653,
                     cp=1.7757,
                     viscosity=0.00007441,
                     molecular_weight_c=26.1782,
                     molecular_weight_ex=27.4389,
                     massflow=2.21,
                     throat_dia=38.63 / 1000,
                     prandtl=0.6821,
                     prop_name="IPA_N2O")

    IPA_LOX = Engine(combustor_temp_K=3325.9758,
                     exit_dia=116.97 / 1000,
```

```
                    combustor_pressure_pa=3000000,
                    gamma=1.1482,
                    k=0.2614,
                    cp=3.3169,
                    viscosity=0.0000883,
                    molecular_weight_c=23.4,
                    molecular_weight_ex=24.9356,
                    massflow=3.47,
                    throat_dia=50.31 / 1000,
                    prandtl=0.595,
                    prop_name="IPA_LOX")

    ETH_N2O = Engine(combustor_temp_K=3204.0548,
                    exit_dia=86.88 / 1000,
                    combustor_pressure_pa=3000000,
                    gamma=1.2148,
                    k=0.1653,
                    cp=1.7757,
                    viscosity=0.00007441,
                    molecular_weight_c=26.1728,
                    molecular_weight_ex=27.4389,
                    massflow=2.21,
                    throat_dia=38.63 / 1000,
                    prandtl=0.6821,
                    prop_name="ETH_N2O")

    RP1_N2O = Engine(combustor_temp_K=3245.2472,
                    exit_dia=87.05 / 1000,
                    combustor_pressure_pa=3000000,
                    gamma=1.2080,
                    k=0.1606,
                    cp=1.7889,
                    viscosity=0.0000752,
                    molecular_weight_c=27.0252,
                    molecular_weight_ex=28.4406,
                    massflow=2.23,
                    throat_dia=38.54 / 1000,
                    prandtl=0.6736,
                    prop_name="RP1_N2O")

    RP1_LOX = Engine(combustor_temp_K=3536.0211,
                    exit_dia=111.29 / 1000,
                    combustor_pressure_pa=3000000,
                    gamma=1.1588,
                    k=0.2574,
                    cp=3.2179,
                    viscosity=0.00008715,
                    molecular_weight_c=22.9155,
                    molecular_weight_ex=24.5798,
                    massflow=3.07,
                    throat_dia=48.23 / 1000,
                    prandtl=0.4933,
                    prop_name="RP1_LOX")

    PRO_LOX = Engine(combustor_temp_K=3484.1915,
                    exit_dia=110.84 / 1000,
                    combustor_pressure_pa=3000000,
                    gamma=1.1617,
                    k=0.2699,
                    cp=3.1793,
                    viscosity=0.00008657,
                    molecular_weight_c=21.7974,
                    molecular_weight_ex=23.2774,
```

```
                    massflow=3,
                    throat_dia=48.11 / 1000,
                    prandtl=0.5134,
                    prop_name="PRO_LOX")

    MET_LOX = Engine(combustor_temp_K=3404.8994,
                    exit_dia=109.71 / 1000,
                    combustor_pressure_pa=3000000,
                    gamma=1.1641,
                    k=0.2812,
                    cp=3.1651,
                    viscosity=0.00008513,
                    molecular_weight_c=20.6749,
                    molecular_weight_ex=21.9536,
                    massflow=2.9,
                    throat_dia=47.67 / 1000,
                    prandtl=0.5542,
                    prop_name="MET_LOX")

    Full_Deflector = Deflector(impingement_angle=10,
                               conductivity=45,
                               density=7850,
                               standoff_distance=(115.49 * 3) / 1000,
                               cp=540,
                               diameter=(381 + 0.5 * (
                                           273.9 - 2 * (9.27))) / 1000)

    Engines = [IPA_LOX, IPA_N2O, ETH_N2O, RP1_LOX, RP1_N2O, PRO_LOX, MET_LOX]

    distance_spread = np.linspace(evans_Startpoint, total_Length,
                                  evans_Length)
    evans_Hs = np.zeros((len(distance_spread), 7))
    evans_Fluxes = evans_Hs
    bartz_Fluxes = np.zeros((7, 2))
    eckert_Fluxes = np.zeros((7, 2))

    for i in range(0, 7):
        Engines[i].Expand(expansion_ratio=1)
        Engines[i].Deflect(Full_Deflector)
        evans_Hs[:, i] = evans_HTC(Engines[i], distance_spread)
        evans_Fluxes[:, i] = np.multiply(evans_Hs[:, i], np.sqrt(
            (T_initial - Engines[i].flat_plate_temp_static) ** 2))

        bartz_Fluxes[i, :] = bartz_HFX(T_initial, Engines[i],
                                       Full_Deflector.dia)
        eckert_Fluxes[i, :] = eckert_HFX(T_initial, Engines[i])

    valid_Output = False
    units = ""
    while not (valid_Output):
        output_Type = input(
            "Enter type of results to output, (F)lux, (T)emperature, "
            "or (H)eat transfer coefficient? "
            "\n").lower()
        if output_Type == "flux" or output_Type == "f":
            valid_Output = True
            units = "kW"
            resultindex = 1
            evans_Results = evans_Fluxes / 1000
            bartz_Results = bartz_Fluxes[resultindex] / 1000
            eckert_Results = eckert_Fluxes[resultindex] / 1000
        elif output_Type == "temperature" or output_Type =="t":
            burn_Valid = False
```

```
            while not (burn_Valid):
                burn_Input = input(
                    "Please enter an integer burn time in seconds: ")
                try:
                    burn_Time = int(burn_Input)
                    burn_Valid = True
                except:
                    warnings.warn("Please enter an integer for burn time")
                    burn_Valid = False

            valid_Output = True
            units = "°C"

            evans_Temps = Full_Deflector.Hotspot(evans_Fluxes, burn_Time)
            bartz_Temps = Full_Deflector.Hotspot(bartz_Fluxes, burn_Time)
            eckert_Temps = Full_Deflector.Hotspot(eckert_Fluxes, burn_Time)

            evans_Results = evans_Temps
            bartz_Results = bartz_Temps
            eckert_Results = eckert_Temps
        elif output_Type == "heat transfer coefficient" or output_Type == "h":
            valid_Output = True
            resultindex = 0
            units = "/"
            evans_Results = evans_Hs
            bartz_Results = bartz_Fluxes[resultindex]
            eckert_Results = eckert_Fluxes[resultindex]
        else:
            print("Invalid command.")

    fig, ax = plt.subplots()
    plt.xlabel("Distance from impingement point, mm")

    if output_Type == "flux" or output_Type == "f":
        plt.ylabel("Heat flux on deflector, " + units)
    elif output_Type == "heat transfer coefficient" or output_Type == "h":
        plt.ylabel("Heat transfer coefficient, " + units)
    else:
        plt.ylabel("Temperature along deflector, " + units)

    deflector_Results = np.ones((array_Length, 7))

    for k in range(0, 7):
        evans_Start_Index = int(evans_Startpoint / step_Distance)
        deflector_Results[evans_Start_Index:(evans_Start_Index + evans_Length),
        k] = (
                deflector_Results[evans_Start_Index:(
                evans_Start_Index + evans_Length), k] * evans_Results[:,k])

        deflector_Results[0:evans_Start_Index + len(
            evans_Results[evans_Results[:, k] > eckert_Results[k]]), k] = \
        eckert_Results[k]

        deflector_Results[(evans_Start_Index + evans_Length) - len(
            evans_Results[evans_Results[:, k] < bartz_Results[k]]):, k] = \
        bartz_Results[k]
        if deflector_Results[array_Length - 1,k] != bartz_Results[k]:
            warnings.warn(
                "Bartz result unused, length of flux "
                " evaluation should be increased" + "For Engine " +
                str(
                    Engines[k].prop_name),
                category=UserWarning)
```

```
        if (np.average(evans_Results[0:20, k]) - eckert_Results[k]) > (
                eckert_Results[k] / 5):
            warnings.warn(
            "Large disagreement between Evans and Eckert detected, "
            "consider changing the evaluation method of the skin friction "
            "coefficient within the eckert_HFX function" + "For Engine " + str(
                    Engines[k].prop_name),
                category=UserWarning)

        distance_list = np.linspace(0, total_Length * 1000, array_Length)
        ax.plot(distance_list, deflector_Results[:, k])

    draw_Graph = input(
        "Draw data graph y/n? \n").lower()
    if draw_Graph == "y":
        plt.show()

    write_CSV = input(
        "Export data as .csv y/n? \n").lower()

    if write_CSV == "y":

        Fields = ["Distance (mm)", "", "", "", "", "", "", "", ""]
        for j in range(1, 8):
            Fields[j] = str(
                "Heat Flux (" + units + ") , " + str(Engines[j-1].prop_name))

        Data = np.column_stack((distance_list, deflector_Results))
        current_time = str(datetime.datetime.now())[:16].replace(
            " ", "-").replace(":", "-")
        with open(str(output_Type) + "_" + current_time + "_MultiEngine.csv",
                  "w",
                  newline="") as f:
            csv_writer = csv.writer(f, dialect="excel")
            csv_writer.writerow(Fields)
            for i in range(0, np.shape(Data)[0]):
                csv_writer.writerow(Data[i, :])

    print("Done! in " + str(datetime.datetime.now() - start_time) + "s")
    return


if __name__ == '__main__':
    Main()
    exit()
```

### B.2.4 Modified code required for more complex flow modelling

```
class Deflector:
    def __init__(self, impingement_angle, conductivity, density,
                 standoff_distance, cp, diameter, inlet_length,
                 curvature_radius = None, outlet_length=None):
        self.impingement_angle = impingement_angle
        self.standoff_distance = standoff_distance
        self.conductivity = conductivity
        self.density = density
        self.cp = cp
        self.dia = diameter
        self.inlet = inlet_length
        if curvature_radius is not None:
```

```
            self.curvature = curvature_radius
        else:
            self.curvature = diameter

        if outlet_length is not None:
            self.outlet = outlet_length
        else:
            self.outlet = self.inlet

def prandtl_meyer(mach, gamma=1.4):
    #Evaluate Prandtl-Meyer function at given Mach number.

    return (
            np.sqrt((gamma + 1) / (gamma - 1)) *
            np.arctan(np.sqrt((gamma - 1) * (mach ** 2 - 1) / (gamma + 1))) -
            np.arctan(np.sqrt(mach ** 2 - 1))
    )


def solve_prandtl_meyer(mach, nu, gamma=1.4):
    #Solve for unknown Mach number, given Prandtl-Meyer function (in
    # radians).

    return (nu - prandtl_meyer(mach, gamma))

def get_rayleigh_reference_temperature(mach, gamma=1.4):
    '''Return T/T* for Rayleigh flow'''
    return (
        mach**2 * (1 + gamma)**2 /
        (1 + gamma*mach**2)**2
        )

def get_rayleigh_reference_stag_temperature(mach, gamma=1.4):
    '''Return Tt/Tt* for Rayleigh flow'''
    return (
        2*(1 + gamma)*mach**2 *
        (1 + 0.5*(gamma - 1)*mach**2) / (1 + gamma*mach**2)**2
        )

def solve_rayleigh_reference_stagnation_temperature(mach, Tt_Ttstar, gamma=1.4):
    '''Used to find unknown Mach number given Tt/Tt* and gamma'''
    return (
        Tt_Ttstar - get_rayleigh_reference_stag_temperature(mach, gamma)
        )

class Engine:
    def __init__(self, combustor_temp_K, exit_dia, combustor_pressure_pa, gamma,
                 k, viscosity, cp, massflow,
                 throat_dia, prandtl, r_comb=None, r_ex=None,
                 molecular_weight_c=None, molecular_weight_ex=None,
                 throat_vel=None):
        self.T_stag = combustor_temp_K
        self.P_stag = combustor_pressure_pa
        self.gamma = gamma
        self.conductivity = k
        self.mu = viscosity
        self.cp = cp
        self.r_c = r_comb
        self.r_ex = r_ex
        self.mw_c = molecular_weight_c
        self.mw_ex = molecular_weight_ex
        self.exit_dia = exit_dia
        self.throat_dia = throat_dia
```

```
        self.throat_vel = throat_vel
        self.v_ex = None
        self.a_ex = None
        self.mdot = massflow
        self.Pr = prandtl

        if [self.mw_c, self.mw_ex] == [self.r_c, self.r_ex]:
            raise Exception(
                "Definition failure, Please specifiy at least " +
                "one molecular weight or value for R\n")

        # account for single mw value
        if self.mw_ex is None:
            self.mw_ex = self.mw_c
        # account for missing r values
        if self.r_c is None:
            self.r_c = (fluids.constants.gas_constant * 1000) / self.mw_c

        if self.r_ex is None:
            self.r_ex = (fluids.constants.gas_constant * 1000) / self.mw_ex

    def Expand(self, expansion_ratio=None, expansion_pressure=None,
               exhaust_mach=None):
        # Nozzle properties
        nozzle_properties = [expansion_ratio, expansion_pressure, exhaust_mach]
        for i in range(0, len(nozzle_properties)):
            if nozzle_properties[i] is not None:
                nozzle_properties[i] = 1
        if nozzle_properties == [0, 0, 0]:
            if self.throat_dia is not None and self.throat_vel is not None:
                area_ratio = self.exit_dia / self.throat_dia
                Ma_ex = np.sqrt((2 / (self.gamma - 1)) * ((area_ratio ** (
                        (self.gamma - 1) / self.gamma)) - 1))
                stag_pressure_ratio = (1 + ((self.gamma - 1) / 2) * (
                        Ma_ex ** 2)) ** (self.gamma / (self.gamma - 1))
                self.Psa_ex = self.P_stag / stag_pressure_ratio
                self.Tsa_e = self.T_stag / (
                        1 + (((self.gamma - 1) / 2) * Ma_ex ** 2))
                self.a_ex = np.sqrt(self.r_ex * self.Tsa_e)
                self.v_ex = self.a_ex * Ma_ex
            else:
                raise Exception(
                    "Specify nozzle properties when calling the Expand function"
                    + ", or exit ratio during engine definition\n")
        match nozzle_properties:
            case [1, None, None]:
                self.Psa_ex = expansion_ratio * 101325
                Ma_ex = np.sqrt((((self.P_stag / self.Psa_ex) ** (
                        (self.gamma - 1) / self.gamma)) - 1) * 2 / (
                                        self.gamma - 1))
                self.Tsa_e = self.T_stag / (
                        1 + (((self.gamma - 1) / 2) * Ma_ex ** 2))
                self.a_ex = np.sqrt(self.r_ex * self.Tsa_e)
                self.v_ex = self.a_ex * Ma_ex
            case [None, 1, None]:
                self.Psa_ex = expansion_pressure
                Ma_ex = np.sqrt((((self.P_stag / self.Psa_ex) ** (
                        (self.gamma - 1) / self.gamma)) - 1) * 2 / (
                                        self.gamma - 1))
                self.Tsa_e = self.T_stag / (
                        1 + (((self.gamma - 1) / 2) * Ma_ex ** 2))
                self.a_ex = np.sqrt(self.r_ex * self.Tsa_e)
                self.v_ex = self.a_ex * Ma_ex
```

```
            case [None, None, 1]:
                Ma_ex = exhaust_mach
                stag_pressure_ratio = (1 + ((self.gamma - 1) / 2) * (
                        Ma_ex ** 2)) ** (self.gamma / (self.gamma - 1))
                self.Psa_ex = self.P_stag / stag_pressure_ratio
                self.Tsa_e = self.T_stag / (
                        1 + (((self.gamma - 1) / 2) * Ma_ex ** 2))
                self.a_ex = np.sqrt(self.r_ex * self.Tsa_e)
                self.v_ex = self.a_ex * Ma_ex
            case _:
                raise Exception("Please specify exhaust properties\n")

    def Propagate(self, distance, plume_dia=None):
        if self.v_ex is None:
            raise Exception(
                "Please use the Expand function before calling Propagate\n")
        if distance < ((self.exit_dia / 2) / 0.053):
            return self.v_ex
        if plume_dia is None:
            plume_dia = self.exit_dia
        v_x = ((10 ** 0.79 - (33 * ((plume_dia ** 2) / distance ** 2)))
               / (distance / plume_dia)) * self.v_ex
        return v_x

    def Deflect(self, D: Deflector):
        standoff_distance = D.standoff_distance
        impingement_angle = D.impingement_angle
        P_sa = self.Psa_ex
        P_sg = self.P_stag

        step_Distance = (0.5/ 1000)  # step setting, m
        deflector_length = D.inlet + D.outlet + ((np.pi * D.dia) / 4)
        array_Length = round(deflector_length / step_Distance)
        v_impingement = self.Propagate(
            standoff_distance)  # impingement after standoff angle,
                                # at centre of plume

        a_ex = np.sqrt(self.r_ex * self.Tsa_e)
        mach_at_impingement = v_impingement / a_ex  # velocity change will be
        # small, so anticipated
        # speed of sound change is low

        sigma = get_strong_weak_oblique(mach_at_impingement, impingement_angle,
                                        self.gamma)[0]
        # gets the weak shock angle for the impingement that's provided by the FD
        freestream_shock_angle = 180 - (
                90 + impingement_angle + sigma)
        # Angle from freestream to shock normal in deg.
        mach_normal_1 = mach_at_impingement * np.cos(np.deg2rad(
            freestream_shock_angle))
        # take the mach speed normal to the shock angle
        mach_tangent = mach_at_impingement * np.sin(
            np.deg2rad(freestream_shock_angle))
        mach_normal_2 = np.sqrt(
            (1 + ((self.gamma - 1) / 2) * mach_normal_1 ** 2) /
            ((self.gamma * mach_normal_1 ** 2) - ((self.gamma - 1) / 2))
        )  # find mach normal to shock angle after oblique shock

        mach_after_impingement = np.sqrt(mach_normal_2 ** 2 + mach_tangent ** 2)

        Tsa_after_oblique = self.Tsa_e * (
                (1 + ((2 * self.gamma) / (self.gamma + 1)) * (
                        mach_normal_1 ** 2 - 1)) *
```

```
                ((2 + (self.gamma - 1) * mach_normal_1 ** 2) / (
                        (self.gamma + 1) * mach_normal_1 ** 2))
        )

        speed_of_sound_after_impingement = np.sqrt(
            self.r_ex * self.gamma * Tsa_after_oblique)
        # speed of sound after temp change through shock

        speed_along_FD = mach_after_impingement * speed_of_sound_after_impingement

        self.flat_plate_speed = speed_along_FD
        self.flat_plate_temp_static = Tsa_after_oblique

        self.velocities = speed_along_FD * np.ones(array_Length)
        self.distances = np.linspace(0, deflector_length, array_Length)
        self.temps_st = Tsa_after_oblique * np.ones(array_Length)
        self.temps_sg = (((1 + ((self.gamma - 1) / 2)) *
                          mach_after_impingement * Tsa_after_oblique) *
                          np.ones(array_Length))
        self.machs = mach_after_impingement * np.ones(array_Length)
        self.a_values = speed_of_sound_after_impingement * np.ones(array_Length)
        self.pres_st = P_sa * np.ones(array_Length)
        self.pres_sg = P_sg * np.ones(array_Length)

        curved_surface = ((np.rad2deg(np.arcsin((self.exit_dia/2)/D.curvature)) *
                    2)/360) * (2*D.curvature * np.pi)
        # arc length of portion of deflector wetted by flow

        self.areas = (np.ones(array_Length) * curved_surface * deflector_length/
                       array_Length)
        # area at each point along the deflector, a = x * arc length
        for i in range(0, array_Length):
            if self.distances[i] >= (self.exit_dia/2):
                break
            else:
                self.areas[i] = 2 * np.sqrt(((self.exit_dia / 2) ** 2) -
                        ((self.exit_dia / 2) - self.distances[i]) ** 2) *(
                        deflector_length/ array_Length)
                # area = with of circle slice at x * x
        return


    def Rayleigh(self, distances, heat_fluxes, start_point, override=False):
        gamma = self.gamma

        if start_point == len(distances):
            return
        for i in range(start_point, len(distances)-start_point - 1):
            # Energy in
            area = self.areas[i-1]
            flux = -1 * heat_fluxes[i-1] # flux is cooling the flow
            dwell_time = ((self.distances[i] - self.distances[i-1])/
                          self.velocities[i])
            temp_change = (area * flux * dwell_time) / self.cp
            self.temps_sg[i] = self.temps_sg[i-1] + temp_change
            ma_1 = self.machs[i-1]
            T_t1 = self.temps_sg[i-1]
            T_t2 = self.temps_sg[i]

            Tt1_Tstar = get_rayleigh_reference_stag_temperature(ma_1, gamma)

            Tt2_Tstar = (T_t2 / T_t1) * Tt1_Tstar

```

```
            root = scipy.optimize.root_scalar(
                solve_rayleigh_reference_stagnation_temperature, x0=1.2, x1=2.4,
                args=(Tt2_Tstar, gamma)
            )
            ma_2 = root.root
            p_st_ratio = (1 + gamma*ma_1**2)/(1 + gamma * ma_2 **2)
            p_sg_ratio = (p_st_ratio * ((1 + ((gamma-1)/2)*ma_1**2)
                          /(1 + ((gamma-1)/2) *ma_2 **2))**(gamma/(gamma - 1)))
            self.pres_st[i] = self.pres_st[i - 1] * p_st_ratio
            self.pres_sg[i] = self.pres_sg[i - 1] * p_sg_ratio
            self.temps_st[i-1] = Tt1_Tstar / (1 + ((gamma - 1)/2)*ma_1**2)
            self.temps_st[i] = Tt2_Tstar / (1 + ((gamma - 1)/2)*ma_2**2)
            self.machs[i] = ma_2
            self.a_values[i] = np.sqrt(gamma * self.r_ex * 1000 *
                                       self.temps_st[i-1])
            self.velocities[i] = self.a_values[i] * self.machs[i-1]
        if override:
            self.a_values[i:] = self.a_values[i-1]
            self.velocities[i:] = self.velocities[i-1]
            self.pres_st[i:] = self.pres_st[i-1]
            self.pres_sg[i:] = self.pres_sg[i-1]
            self.machs[i:] =self.machs[i-1]
        return


    def PrandtlMeyer(self, D:Deflector, override=False):
        gamma = self.gamma

        quarter_circumference = (0.25*np.pi * D.dia)
        step_size = (D.inlet + D.outlet + quarter_circumference) / len(
            self.distances)
        start_index = round(D.inlet / step_size) - 1
        arc_length = quarter_circumference / step_size
        angle_step = 90 / round(arc_length)
        angle = 0
        ma_2 = ma_1 = self.machs[start_index]

        for i in range(0, round(arc_length / quarter_circumference)):
            angle += angle_step
            current_index = start_index + i

            delta_nu = np.deg2rad(-angle_step)
            nu_1 = prandtl_meyer(ma_1, gamma)
            nu_2 = nu_1 + delta_nu

            if nu_2 < np.deg2rad(1) or np.isnan(nu_1):
                break

            root = scipy.optimize.root_scalar(solve_prandtl_meyer, x0=ma_2 -
                                                  0.1,x1=ma_1,
                                              args=(nu_2, gamma))
            ma_2 = root.root
            self.machs[current_index] = ma_2
            self.temps_st[current_index] = (self.temps_st[current_index - 1] *
                                            (1 + ((gamma - 1)/2 * ma_1**2))/
                                            (1 + ((gamma - 1) / 2 * ma_2**2)))

            self.temps_sg[current_index] = (self.temps_st[current_index] *
                                            (1+((gamma - 1)/2 * ma_2**2)))

            self.a_values[current_index] = np.sqrt(gamma * self.r_ex * 1000 *
                                                   self.temps_sg[current_index])
            self.velocities[current_index] = self.machs[current_index] * (
```

```
                self.a_values[
                current_index])
            p_st_ratio = (1 + gamma*ma_1**2)/(1 + gamma * ma_2 **2)
            p_sg_ratio = (p_st_ratio * ((1 + ((gamma-1)/2)*ma_1**2)
                         /(1 + ((gamma-1)/2) *ma_2 **2))**(gamma/(gamma - 1)))
            self.pres_st[current_index] = self.pres_st[
                                                current_index - 1] * p_st_ratio
            self.pres_sg[current_index] = self.pres_sg[
                                                current_index - 1] * p_sg_ratio
            ma_1 = ma_2
        if override:
            self.a_values[current_index:] = self.a_values[current_index-1]
            self.velocities[current_index:] = self.velocities[current_index-1]
            self.pres_st[current_index:] = self.pres_st[current_index-1]
            self.pres_sg[current_index:] = self.pres_sg[current_index-1]
            self.machs[current_index:] =self.machs[current_index-1]

        return

def evans_HTC(E: Engine, distance_from_impingement_m, T_sa, P_sa,
              velocity=None):
    # convert all units to us customary
    cp = E.cp * 1000 * 2.402
    # cp BTU/ lb °F
    conductivity = E.conductivity * 0.5781759824
    # conductivity Btu (th) foot/ hour ft^2 °F
    density = (P_sa * 0.062427962) / (E.r_ex * T_sa)
    # density lb/ft3
    viscosity = E.mu * 17.8579688
    # dynamic viscosity lb/ in s
    if velocity is not None:
        velocity = velocity * 3.280839895
    else:
        velocity = E.flat_plate_speed * 3.280839895
    # velocity ft/s
    distance_from_impingement = distance_from_impingement_m * 3.280839895
    # distance ft

    heat_transfer_coefficient_imperial = ((0.0296 *
        (((density * velocity * distance_from_impingement) / viscosity) ** (
                            4 / 5))* (((cp * viscosity) / conductivity) ** (
                            1 / 3)) * conductivity)/ distance_from_impingement)
    # heat transfer coefficient btu / in^2 sec °F

    heat_transfer_coefficient_metric = (
            heat_transfer_coefficient_imperial * 8.611128)
    return heat_transfer_coefficient_metric


def bartz_HFX(T_Wall, E: Engine, T_sa):
    cp = E.cp * 1000 * 0.2390057361
    # cp BTU / lb °F
    viscosity = E.mu * 0.000145
    # viscocity lbf sec / in
    throat_Dia = E.throat_dia * 39.3701
    # throat diameter in
    Pr = E.Pr
    T_sa = (T_sa) * (9 / 5)
    # exhaust temperature °R
    T_Wall = (T_Wall) * (9 / 5)
    # wall temperature °R
    Ma = E.v_ex / E.a_ex
    Cstar = ((E.P_stag * (
```

```
            np.pi * (E.throat_dia / 2) ** 2)) / E.mdot) * 3.280839895
    # C* ft/s
    g = 32.2
    # gravitation acceleration ft/s^2
    PC = E.P_stag * 0.000145038
    # Chamber Pressure lbf / in^2
    gamma = E.gamma
    exit_Dia = E.exit_dia * 39.3701
    # Exit Diameter in

    sigma = ((((T_Wall / (2 * T_sa)) * (
            1 + ((gamma - 1) / 2 * Ma ** 2))) + 0.5) ** (-0.68) * (
                    (1 + ((gamma - 1) / 2 * Ma ** 2)) ** (-0.12)))

    T_effective = T_sa * ((1 + ((Pr ** 0.33) * ((gamma - 1) / 2 * Ma ** 2))) / (
            1 + ((gamma - 1) / 2 * Ma ** 2)))

    heat_transfer_coefficient = ((((0.026) / (throat_Dia ** 0.2)) * (
            (viscosity ** 0.2 * cp) / (Pr ** 0.6)) *
                        ((PC * g) / Cstar) ** 0.8) *
                       ((((throat_Dia / 2) ** 2) * np.pi) /
                        (((exit_Dia / 2) ** 2) * np.pi) ** 0.9) * sigma)
    # heat transfer coefficient BTU/ in sec °F

    factor = (11356.526682 / 12)
    heat_flux_metric = factor * (heat_transfer_coefficient) * (
        abs(T_effective - T_Wall))

    return heat_flux_metric


def eckert_HFX(T_wall, E: Engine, T_sa, T_sg, P_sa, V):
    Pr = E.Pr
    Cp = E.cp * 1000
    rho = (P_sa) / (E.r_ex * T_sg)
    mu = E.mu
    Re = (rho * V) / mu


    Cf = 0.0296 / (Re ** (0.2))
    # lower values, Blasius equation
    # Cf = 0.37/((np.log10(Re))**2.53)
    # higher values of heat flux, schlutz-grunow

    r = Pr ** (1 / 3)
    T_r = (r * abs(T_sg - T_sa)) + T_sa
    St = (Cf / 2) * (Pr ** -(2 / 3))

    h = St * (rho * Cp * V)
    flux = h * abs(T_r - T_wall)
    return flux
```