Spring 2008

Addressing nonlinear combustion instabilities in highly dilute spark ignition engine operation

Brian C. Kaul

Department: Mechanical and Aerospace Engineering

ADDRESSING NONLINEAR COMBUSTION INSTABILITIES IN HIGHLY DILUTE SPARK IGNITION ENGINE OPERATION

by

BRIAN CHRISTOPHER KAUL

A DISSERTATION

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

in

MECHANICAL ENGINEERING

2008

Approved by

James A. Drallmeier, Advisor
Kakkattukuzhy M. Isaac
Robert G. Landers
David W. Riggins
Jagannathan Sarangapani

ABSTRACT

Dilute operation is a promising approach for increasing spark-ignition engine efficiency, in the form of either lean burn (air dilution) or EGR (inert dilution). High levels of charge dilution, however, lead to cyclic variability that is largely deterministic in nature. The determinism and nonlinear nature of the system indicate that it should be possible to reduce the cycle-to-cycle variations by implementing an electronic controller. Several needs arise when considering the development of such a controller.

Three topics of interest are covered herein. First, a method of analysis for nonlinear dynamical systems is applied to engine data in order to estimate the effect that a controller could have by removing the cycles that contribute to repeated, deterministic sequences. Among other things, this allows for determination of when controlled, highly dilute SI engine operation would be a desirable combustion mode.

Second, the sensitivity of the engine to variations in control input is evaluated by examining a FFT of heat release data when the injected fuel mass is perturbed in a periodic manner. When the amplitude of variations is sufficient to effect discernable change in the engine behavior, variations of the imposed frequency are apparent in the heat release sequence. The engine was found to be more sensitive to changes in control input at higher dilution levels.

Finally, a combined thermodynamic and turbulent mass entrainment model was developed to predict energy release for many consecutive engine cycles. This model captures the cyclic dynamics of actual engine behavior, based on physics rather than arbitrary mathematical functions. It should therefore be useful in future controller development and simulations.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. James Drallmeier, for the opportunity to learn from him and for his advice and direction throughout the course of my graduate studies. Also, thanks to the members of my committee, Drs. Sarangapani, Riggins, Landers, and Isaac, for their assistance and input.

I am also grateful to Jonathan Vance, Peter Shih, and the other electrical engineering students who worked on the controls aspect of the research for their assistance in the lab, and to Dr. Robert M. Wagner of the Oak Ridge National Laboratory for his input and suggestions regarding a symbolic approach to data analysis.

This work would not have been possible without funding support from the National Science Foundation through ECCS#0327877 and from the GAANN Fellowship from the Department of Education.

Thanks also to my fellow students in the internal combustion engine lab, and others who have been friends and colleagues during my time in Rolla. Finally, thanks to my friends and family, for their support over the years.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

NOMENCLATURE

Symbol	Description
y	Output parameter (heat release)
γ	Specific heat ratio
p	Pressure (in-cylinder)
θ	Crankshaft angle
V	Cylinder volume
V_C	Clearance volume
r	Compression ratio
R	Rod-crank ratio (cylinder volume calculation)
m_a	Mass of air
k	Cycle index (Daw model)
F_{res}	Residual fraction
$\left(\frac{A}{F}\right)_S$	Stoichiometric air/fuel ratio
CE	Combustion efficiency
m_f	Mass of fuel
φ	Equivalence ratio
φ_u	Upper equivalence ratio parameter (Daw model)
φ_l	Lower equivalence ratio parameter (Daw model)
φ_m	Mean equivalence ratio parameter (Daw model)
$\left(\frac{F}{A}\right)_A$	Actual fuel/air ratio
$\left(\frac{F}{A}\right)_S$	Stoichiometric fuel/air ratio
α	Constant of proportionality (Daw model output)
F_b	Baseline frequency
n_{part}	Number of partitions
l_{seq}	Sequence length
H_S	Shannon entropy (modified)
p_k	Probability of occurrence of sequence k

$\dot{Q}$	Cylinder wall heat transfer rate
A_h	Surface area of boundary between cylinder wall and burned zone
h	Heat transfer coefficient (wall heat transfer calculation)
T_w	Cylinder wall temperature
T_{avg}	Average charge temperature
L	Stroke
N	Engine speed
λ	Relaxation factor
m_u	Mass of unburned gases
R_u	Gas constant of unburned gases
T_u	Temperature of unburned gases
m_b	Mass of burned gases
R_b	Gas constant of burned gases
T_b	Temperature of burned gases
$\frac{dm_e}{dt}$	Rate of change of entrained mass
ρ_u	Density of unburned gases
A_f	Flame surface area
u'	Turbulence intensity
S_L	Laminar flame speed
ρ	Flame radius (flame geometry calculations)
V_b	Volume of burned zone
R	Cylinder radius (flame geometry calculations)
h	Cylinder height (flame geometry calculations)
$\frac{dm_b}{dt}$	Rate of change of burned mass
m_e	Mass of unburned gases entrained in flame
τ	Characteristic burning time
l_m	Taylor microscale
l_i	Turbulence integral scale
ρ_0	Unburned gas density at start of combustion
$(l_i)_0$	Turbulence integral scale at start of combustion

$(u')_0$	Turbulence intensity at start of combustion
ν	Dynamic viscosity
ν_S	Dynamic viscosity at standard conditions
ρ_S	Unburned gas density at standard conditions
T_S	Standard temperature
S_{L_0}	Reference laminar flame speed
α	Temperature constant (laminar flame speed variation)
β	Pressure constant (laminar flame speed variation)
Y_{dil}	Diluent concentration
B_M	Fuel constant 1 (reference laminar flame speed)
B_2	Fuel constant 2 (reference laminar flame speed)
φ_M	Fuel constant 3 (reference laminar flame speed)

1. INTRODUCTION

1.1. BACKGROUND/MOTIVATION

Rising fuel prices and regulations aimed at reducing carbon dioxide (CO_2) emissions on the part of some regulatory agencies make it desirable to improve the efficiency of internal combustion engines used in both the automotive and heavy equipment industries. Diesel, or compression ignition (CI), engines have long been favored for heavy equipment and over-the-road trucking applications due to their higher efficiency and durability compared to spark ignition (SI) engines. Future EPA standards, however, require substantial reductions in oxides of nitrogen (NO_x) and particulate emissions, necessitating costly particulate filters and lean NO_x catalysts. SI engines, on the other hand, have negligible particulate emissions, and make use of the well-established three-way catalytic converter to control unburned hydrocarbons (HC), NO_x, and carbon monoxide (CO) emissions, yet due to higher pumping losses and lower compression ratios, lack the efficiency of Diesels. The higher combustion temperatures in SI engines also contribute to reduced durability.

Operation at highly dilute conditions is one potential path to significant increases in fuel efficiency while using current technology catalytic systems for emission reduction. Both lean (excess air) dilution and exhaust gas recirculation (EGR), or inert dilution, offer similar benefits and pitfalls. A high dilution SI engine, or mixed-mode SI/dilute SI/homogeneous charge compression ignition (HCCI) engine would be a possible lower cost alternative to Diesels for future heavy equipment emissions standards, since it would not require the costly after-treatment devices and highly dilute operation could make the gasoline engine comparable to the Diesel in efficiency and durability. Under these highly dilute conditions, however, combustion becomes "strained," and large cycle-to-cycle variations in engine output are encountered. To reach the full potential of load control through dilution, the dispersion in cyclic output under highly dilute conditions must be addressed.

Once the issues of cyclic dispersion in output are addressed, significant efficiency gains can be made with dilute SI combustion. Fuel efficiency will be improved due to lower pumping losses, along with a corresponding reduction in CO_2 emissions. Since

inert dilution via EGR allows for stoichiometric operation, current three-way catalysts can be used for emissions control without requiring the costly particulate traps necessary for Diesels, and since the operation is still in an SI mode, unlike HCCI, precise control over the start of combustion is still available. High dilution rates will also lead to reduced full-load octane requirements due to lower knock tendencies, as well as increased durability over non-dilute homogeneous SI due to reduced temperatures. Additionally, this approach is fuel-flexible, so bio-fuels can be used in place of fossil fuels if the market demands. These benefits mark dilute SI combustion as a particularly promising approach.

However, progress still needs to be made in extending the dilution limits of homogeneous SI combustion. In order to eliminate the problematic cycle-to-cycle variations in engine output under these conditions, new control schemes must be developed. Several specific issues relevant to this goal are addressed herein. First, Chapter **Error! Reference source not found.** will address new applications of data analysis techniques that allow for estimates of the effectiveness of a controller at various operating conditions as well as some insight into controller design. Second, Chapter 4 describes experiments that show the sensitivity of the engine output to changes in control inputs under the highly dilute operating conditions that are of interest. Third, Chapter 5 describes the development of a multi-cycle engine model that includes the relevant combustion physics and that will allow for more informed design of future control schemes and for simulations of controllers in software prior to implementation on actual engines. Chapter 2 describes the laboratory setup that was used for experimental data collection, and Chapter 6 summarizes the conclusions. In the remainder of this chapter, a review of relevant literature is presented.

1.2. REVIEW OF LITERATURE

1.2.1. Approaches to Improving SI Engine Emissions and Efficiency.

Numerous approaches exist to the problem of how best to improve the efficiency of SI engines. Turner, et al. (1) reviewed many such approaches in 2004 and offered their suggestions for a map towards meeting future legislative requirements for CO_2 emissions. They point out that reducing vehicle mass yields only about one-half of a percentage return in fuel economy benefits for each percentage reduction in mass, and improvements

in aerodynamics offer only about a 1:4 return, while improvements to powertrain efficiency offer 1:1 returns, and thus they focus on improving SI engine efficiency.

One class of improvements described is advanced valve train systems, including both mechanical variable valve timing (MVVT) and fully variable valve timing (FVVT), as well as cylinder deactivation (CDA). CDA is effectively on-the-fly downsizing of the engine, allowing the active cylinders to operate at a higher load and efficiency. Emissions of HC and CO are reduced due to the reduced crevice volume and higher load operation. NO_x can be reduced if EGR is used, but can be worse without EGR due to the higher load (and thus temperatures) in the operating cylinders. Noise, vibration, and harshness (NVH) can be problematic for engines with less than eight cylinders and necessitate higher idle speeds, but they do report improvements of 7-15% in fuel economy over regulatory test cycles for four-cylinder test vehicles.

MVVT systems are said to often utilize camshaft phasing, and modify the lift and duration of the valves as a means of load control. FVVT systems being developed often utilize electro-hydraulic actuators in place of camshafts and allow full control of the lift, duration, and timing of each valve, and may be either open-loop or closed-loop in their control strategies. These systems allow for advanced combustion modes such as HCCI to be used, either in all cylinders or selected cylinders.

Another topic described by Turner, et al. (1) is gasoline direct injection (GDI), which allows for stratified charge lean SI operation. Both wall-guided and spray-guided GDI operation are discussed. Lean operation has the disadvantage of requiring costly NO_x aftertreatment devices, rather than being able to use three-way catalysts, and early production wall-guided GDI engines are said to have offered disappointing efficiency gains due to inability to eliminate throttling losses. The authors predict that homogeneous spray-guided GDI operation, combined with VVT systems, will be the "cost-effective route forwards ensuring global fuel compatibility."

Others have considered additional aspects of stratified dilute operation. Zhao, et al. (2), compare stratified EGR to "stratified air" (lean) dilution. They determined that while stratified charge lean operation yields greater efficiency gains, stratified EGR offers superior suppression of NO formation. They also found a much lower dilution limit for EGR due to its effect on flame speed as well as its higher heat capacity, but

found that the dilution gradient has little effect on combustion or NO formation as long as the dilution level within the mixing region is within the combustible dilution limit.

The conclusions with respect to the effects of EGR versus lean operation are echoed in Quader, et al. (3) in a study evaluating H_2 addition to dilute engine operation near the dilute limit. The H_2 reformate was found to significantly improve both combustion initiation and burn rates at equivalent dilution levels, allowing for more robust engine operation at operating points near the dilute limit, and a dramatic reduction in engine-out NO_x emissions. Smith and Bartley (4) similarly used synthetic gas as an additive to extend the EGR limit in a natural gas engine.

Stratified charge operation has also been examined for port fuel injected (PFI) engines. The effect of injection timing in a PFI SI engine was studied by Ohm, et al. (5). They report that injection timing can have a strong effect on the lean misfire limit and combustion stability if fuel is injected during the intake stroke. Injection towards the end of the intake stroke can create a stratified charge, similar to late GDI injection. They also found that the distribution of the fuel due to swirl is important for such stratification.

Another charge stratification method, described by Tabata, et al. (6), is to use mixture injection. Here, a premixed fuel/air mixture is injected into the cylinder separate from the inducted air. The stratification of this pre-mixed mixture in the cylinder allows for much higher levels of EGR than would otherwise be possible. The authors report a dilution limit of 48% EGR.

Engine downsizing is another method for improving efficiency discussed by Turner et al. (1), wherein a smaller displacement engine, which is running at a higher load under normal operation is used. Techniques listed for doing this without reducing performance unacceptably include variable compression ratio engines (reduced compression at full load reduces the likelihood of auto-ignition, which can damage engines). Compression ratio can be varied by changing the piston/crank geometry, and another way of achieving the same effect is said to be a late intake valve closing (thus rejecting some of the intake charge and starting compression later). Charge dilution in conjunction with forced induction is also discussed, along with other issues relative to forced induction. Efficiency gains of around 12% are reported for such methods.

De Petris, et al. (7) evaluated high EGR operation in high compression ratio (up to r = 13.5) SI engines. They concluded that EGR has the ability to improve the knock tendency of stoichiometric mixtures and to reduce NO_x emissions. This allows for the use of a higher compression ratio, thereby increasing the thermodynamic efficiency. The authors also report efficiency gains greater than 10% for the same IMEP.

Topinka, et al. (8) performed experiments comparing the knock characteristics of a lean-burn engine with a reference fuel blend of isooctane and n-heptane to the characteristics with H_2 and CO enhanced mixtures, simulating the effects of a portion of the fuel being processed in a plasmatron fuel reformer. They found that knock susceptibility is not reduced for lean operation at a fixed load (it is reduced with decreasing equivalence ratio, but so is the load). With H_2 and CO, however, knock susceptibility is reduced, as a lower octane number reference fuel is required to obtain knock. They also compare a chemical kinetic model for isooctane combustion to the experimental results to explain that H_2 and CO are effective in increasing the ignition delay, and therefore at reducing the knock tendency.

Lastly, Turner, et al. (1) recount various methods of hybridization, including both series and parallel hybrid electric vehicles and also both hydraulic and pneumatic hybridization. These methods could be used in conjunction with any of the aforementioned methods of improving SI engine efficiency. A timeline is presented of the order in which the authors expect such technology to be adopted, with a progression from mild downsizing to homogeneous GDI operation and cam phasing to reduce part-load throttling losses, then more advanced charging systems, HCCI, and FVVT systems.

The simplest of these methods of improving SI engine efficiency to implement, and one that can be combined with any of the other methods, which require changes to engine hardware, is that of charge dilution, either with excess air or with burned gases. The reasons for the observed increase in efficiency with moderate dilution levels are summarized by Heywood (9) as reduced pumping work as dilution is increased at a given load, reduced heat loss to the walls due to the significantly lower burned gas temperature, and a reduction in the degree of dissociation of burned gases allowing more of the fuel's chemical energy to be converted to sensible energy near TDC. Highly dilute SI

combustion is also being pursued by industry groups such as the HEDGE consortium at Southwest Research Institute (10).

1.2.2. Dilute SI Engine Operation. Much work has been done characterizing the problems encountered with high dilution levels in homogeneous SI engine operation. Heywood (9) attributes the eventual drop in efficiency beyond a certain lean equivalence ratio threshold to cycle-to-cycle variations and the increased burn duration of lean mixtures.

1.2.2.1 Dilute Operation Limit. Quader (11) studied the lean limit for SI engines to determine whether flame initiation or flame propagation is the limiting factor. Performing experiments using a single-cylinder CFR engine, he found that spark timing has a significant qualitative effect. For different spark timings, the lean limit will not only be different in magnitude, but can be qualitatively different. For more advanced spark timings an ignition limit is encountered, beyond which some cycles will misfire. For more retarded spark timings a partial burn limit is encountered, beyond which some cycles will not reach complete combustion. For sufficiently lean mixtures, the operation enters a regime where either misfires or partial burns are possible, as both limits have been passed. He concludes that for lean mixtures either flame initiation or flame propagation could be the limiting factor, and that both should be considered in order to identify which is more critical for the desired operating conditions.

The ignition limit is dependent on early flame kernel initiation and growth. The effect of spark power on flame kernel growth has been reported by Cho, et al. (12). Their experiments were carried out with a propane/air mixture at $\varphi = 0.93$ in a turbulent flow system with three different nitrogen dilution levels. Three different ignition systems were compared: a GM High Energy Ignition (HEI) system, a low power breakdown (LPBD) system, and a high power breakdown (HPBD) system. Images of the flame kernel during the different stages of the spark event, including both breakdown and glow discharge phases, were obtained via laser shadowgraphy, and flame radius measurements were averaged over ten events. Their results do not capture the blast wave phase of flame kernel growth, which lasts less than 10 μs, but extrapolate backwards to $t = 0$ to achieve an initial flame kernel size.

They determine that the initial flame kernel size and ionization both increase with increasing breakdown energy. The effects of breakdown energy on the temperature and composition of the early flame kernel are also said to affect its subsequent growth following the blast wave phase. By comparing the different ignition systems, which deliver spark energy at different rates, the authors determine that the energy delivered initially during the breakdown phase has a much greater effect, so a short duration spark produces a faster growing flame kernel than a long duration spark of equal energy. It is also reported that neither turbulence nor dilution affects the initial flame kernel size (immediately following the blast wave phase), though turbulence does increase flame kernel growth rate in later phases, while dilution reduces it. They further observed that the misfire rate, which increased with increasing dilution and turbulence, further increased when changing from long duration glow discharge (HEI) to a short duration breakdown discharge (LPBD) system, with the equivalent total energy held constant, but that increasing the spark power with the short duration breakdown discharge (HPBD) system resulted in no misfires under the conditions tested.

A related study was carried out by Cho and Santavicca (13), determining the effect of mixture inhomogeneity on flame kernel growth. The same turbulent propane/air combustion is studied, but varying the consistency of the fuel/air mixture rather than the spark power. They determined the flow characteristics via LDV, and obtained flame kernel images via high-speed laser shadowgraphy. They found similar average flame kernel growth rates for all but the worst case, which had 33% RMS fluctuations in the fuel/air ratio, but found that cyclic variations increased for incomplete mixing. The misfire rate is also said to increase with mixture inhomogeneity. The results of both of the above flame kernel growth studies are also published in a final DOE report (14).

Cyclic variations for near-stoichiometric conditions as a function of mixture inhomogeneity in a PFI engine due to incomplete mixing were examined using PLIF techniques by Johansson, et al. (15). They report that injection timing choices that cause high levels of inhomogeneity in the vicinity of the spark plug yield unstable engine operation. Modification of the combustion chamber geometry to induce higher levels of turbulence, leading to better mixing and low levels of inhomogeneity regardless of

injection timing eliminated the correlation between fluctuations of fuel/air ratio near the spark plug and combustion quality.

Aleiferis, et al. (16) used chemiluminescence methods to study the relationship between in-cylinder air-fuel ratio and the crank angle at which 5% of the charge has burned in a lean-burn stratified-charge SI engine. Near stoichiometric conditions, they report little correlation. For lean equivalence ratios, however, they found a significant correlation between these parameters, indicating that cyclic combustion variations early in combustion are due, at least in part, to variations in the local mixture composition near the spark plug.

Lawes, et al. (17) studied the variation of turbulent burning rate with equivalence ratios over a range of 0.6 – 2.0 for several different fuels. They used high-speed Schlieren photography and transient pressure measurements to determine a turbulent burning velocity. Turbulent and laminar burning velocities were compared for the same initial temperature, pressure, and equivalence ratio. For methane and methanol, the turbulent burning velocities are reported to have followed the same trends as the laminar burning velocities, as expected. For isooctane, however, fuel-rich turbulent burning velocities are reported to have remained high rather than falling off as quickly. The authors suggest that this might have relevance for stratified lean-burn GDI engines. It appears, though, that lean operation followed the expected trends.

The advantages and disadvantages of lean burn and EGR dilution were also compared and summarized by Lumsden, et al. (18). They show better combustion stability for EGR, and a 90 – 95% reduction in NO_x emissions compared to stoichiometric, non-dilute operation, and more moderate NO_x reductions but better efficiency for lean operation. Lean operation gave a best case specific fuel consumption between 3 and 10% lower than the best case for EGR.

The effect of the composition of diluent has been studied as well. Landman, et al. (19) compared the effects of dilution on NO_x emissions using N_2 and H_2O as diluents in a lean, premixed, turbulent natural gas flame. They found that for all cases, reductions in NO_x emissions were significant compared to dry air only, for both nitrogen and water. Water was also found to have an effect greater than that of nitrogen by a factor of 2 even for a fixed adiabatic flame temperature. The authors conclude from this that water

dilution has effects on NO_x emissions not only through oxygen deficiency and flame temperature reduction, but also through chemical action. This fits with other results showing that dilution with EGR results in a greater suppression of NO formation than dilution with excess air, and also implies that there will be differences between combustion with actual EGR and with simulated EGR using bottled nitrogen gas.

The practical dilution limit is determined by the level of cyclic variability encountered, as more cycles fall into the partial burn regime. Misfires should be avoided altogether, due to their affect on catalytic converters. Several studies have examined the causes of this cyclic variability. Ozdor, et al. (20) experimentally studied several possible causes, concentrating mostly on the spark plug system. They found that even for motored conditions, there was a coefficient of variation (COV) in the peak pressure of 0.5 – 2%, indicating a contribution of valve and ring leakage to the total cyclic variability observed. When the engine was fired, they saw a COV of peak pressure in the range of 6 – 12% for a stoichiometric mixture. Deviations of $\lambda \pm 0.2$ from stoichiometric were found to increase the COV of peak pressure by about 5%. Observed COV of IMEP was reported to be much lower, while the maximum pressure gradient was reported to show much greater variations and to be highly correlated to the peak pressure. The authors determined that cyclic variability is lowest for MBT timing, but is more sensitive to retarded than to advanced timing. They conclude that advanced timing is preferable for reduction of cyclic variability.

The affects of augmenting the spark energy during the glow discharge phase and of changing the spark plug design and orientation are also reported in Ozdor, et al. (20). They determined that changes in spark plug orientation and design cause significant changes in cyclic variability, but that increasing the spark duration and current do not affect the cyclic variability.

The effects of residual gases in the cylinder have also been examined, though primarily for stoichiometric operation. Liu and Karim (21) performed experiments to determine the effect of residual gases on the combustion process in a gas-fueled (propane/methane) engine, with a focus on the autoignition characteristics. They determined that the residual gases from a partial burn cycle have strong kinetic effects, but negligible thermal or diluting effects. Residual gases from complete combustion

cycles following autoignition tended to have significant effects in all three categories. The authors conclude that the build-up in the concentrations of active species from the residual gases over consecutive cycles, as well as the associated thermal effects, contribute to the observed cyclic variations in the onset of autoignition for homogeneous charge, gas-fueled engines.

Juhlin, et al. (22) used planar laser-induced fluorescence of water vapor to measure the residual gas concentration in SI engines. They measured the concentration of water vapor in the spark plug gap to determine the residual level, and found that the residual content close to the spark plug has a strong correlation with cycle-to-cycle variations in the combustion rate. A correlation coefficient of 0.7 is reported for results from a stand-alone combustion chamber, while a correlation coefficient of 0.6 is reported for SI engine operation. They also found that the end of combustion plays a significant role in the residual gas concentration in the next cycle.

1.2.2.2 Consideration of Nonlinear Dynamics and Chaos. The above studies of the effects of the residual gases on the details of the combustion process were focused on near-stoichiometric operation. However, another approach to evaluation of cyclic variability for dilute operation has also indicated a dependence on residual gas composition. Rather than looking only at the effects of parameters in a given engine cycle, it is also possible to consider the time correlation of multiple cycles. As far back as Kantor (23) in 1984, the possibility of chaotic behavior in SI engine operation has been postulated. If chaotic or nonlinear dynamical behavior is occurring, then prediction of, and correction for, upcoming combustion variations should be possible.

Finney (24) applied methods of time series analysis, including peak pressure, pressure rise, and IMEP to various measurable parameters from engine data for near-idle and lean operation. He found that temporal coupling between consecutive engine cycles was difficult to observe in autocorrelation and mutual information functions of peak pressures, but that Kolmogorov entropy, a multidimensional estimator, shows a good degree of serial correlation. A strong temporal correlation for engine speed oscillations was also reported.

Wagner, et al. (25) further studied the origin of cyclic variations in combustion heat release for lean operation from a perspective of nonlinear dynamics. They found

that while variations near stoichiometric conditions are stochastic, as the mixture equivalence ratio is reduced, there is a qualitative transition to nonlinear deterministic behavior via a period-doubling bifurcation sequence. A means of communication between successive cycles is reported to be the residual gas composition. The authors also compare their experimental results with a simple low-order nonlinear dynamical model developed by Daw, et al. (26; 27) and report good agreement, further illustrating the deterministic nature of the cyclic variability.

The same authors also investigated the effects of swirl and fuel injection timing on cyclic variations at lean conditions (28). They found that the equivalence ratio at which deterministic effects become important was strongly influenced by both swirl and fuel injection timing. Qualitatively, the same behavior occurred, but the equivalence ratio at which the transition took place was shifted, depending on the quality of the mixing. This is to be expected, considering the earlier reports of reduced cyclic variability with improved mixture homogeneity (15).

A further study, considering not only changes in swirl and injection timing on one engine, but different engines with substantially different engine designs was later published by Wagner, et al. (29). Here, the authors showed that the transition to deterministic behavior occurs as the equivalence ratio is reduced in both cases. They suggest that considering this result, the underlying cycle dynamics "may not be dependent on the details of such processes as mixing and combustion, but are characteristic of all lean premixed spark ignition engines."

The deterministic nature of the cyclic variability implies the possibility of controlling these variations. While the increased burn duration noted by Heywood (9) will always place an upper limit on the efficiency gains possible due to dilution, elimination of the cycle-to-cycle variations in engine output would significantly extend the practical dilution limit, and allow for load control with charge dilution over a wide range of operating conditions.

1.2.3. Engine Control Approaches. Previous work has examined possible approaches to various engine control problems. Much of this is not relevant to the elimination or reduction of cyclic variations in heat release at dilute operating conditions, but some studies have been done in that area. A summary of those is presented here.

1.2.3.1 Nonlinear Dynamical Approaches. In his doctoral dissertation, van Goor (30) considered the effect of parametric noise on the control algorithm for chaotic dynamical systems that had been published by Ott, et al. (31), termed the "OGY method." This is of interest here because the stochastic components of cyclic variability in dilute SI engine operation can be considered to be parametric noise on the underlying nonlinear dynamical system that causes deterministic cycle-to-cycle variations. Van Goor found that the OGY method, which performs well for "clean" chaotic systems, is "less than optimal" when parametric noise is introduced. Another method, based on the singular value decomposition (SVD) was also tested, and while slightly more robust than the OGY method in noisy environments, was also determined to be suboptimal.

An application of a control strategy that takes advantage of the nonlinearity of flame speed near the lean combustion limit was published by Edwards, et al. (32). They applied a control system to a pulsed combustor operating near the lean limit. The controller would inject a supplemental fuel pulse based on measured pressure fluctuations, and was able to stabilize combustion and extend the stable operation range to leaner equivalence ratios. While the steady flow combustion in a pulsed combustor is certainly different from the discrete combustion events in a reciprocating engine, the application of nonlinear control methods to lean combustion stability is noteworthy.

Applications of such control methods to lean-burn SI engines have also been studied to some degree. Davis, et al. (33) describe a model-based approach to cycle-by-cycle control via fuel pulse width variation of a 4.6L V8 engine based on measured crankshaft accelerations. They report as much as a 30% reduction in RMS variation near the lean limit. This method relies on advanced knowledge of the dynamics encountered to predict and modify each combustion event. The control method used is also described in a US Patent (34).

Another nonlinear dynamical application to SI engine control was presented by Green, et al. (35). There, two approaches were used. The first is to symbolize the data, representing combustion feedback data as discrete symbols, and analyzing the time sequence based previously developed data analysis approaches (27; 36; 37). The symbolically analyzed data was matched to a "library" of previously generated model scenarios to predict future combustion events on a cyclic basis, and control them. The

other approach presented was to reconstruct the underlying dynamical map function from polynomial fits to combustion feedback data using the approach described by Wagner, et al. (38). They evaluated both methods using the model of Daw, et al. (26), and found that both strategies are effective at reducing cyclic combustion variations in the model. The authors suggest that a combination of the two strategies would further improve the ability to control cyclic dispersion.

1.2.3.2 Artificial Intelligence Approaches. The aforementioned approaches to control are all either model-based, requiring a simple yet accurate online model of the engine dynamics in a particular operating condition, or library-based, requiring a catalog of previously determined operating maps to which the engine behavior can be compared. This is not prohibitive for any given engine operating condition, but since the return map of the cyclic dynamics changes both quantitatively and qualitatively when parameters are varied, substantial time must be allocated to tuning the models for a large number of different operating conditions, or else building a large library of maps offline for these varied operating conditions. Furthermore, this tuning must be repeated for each engine to which the control method would be applied. It would be desirable, then, to have a control method that does not require an involved tuning process for every application, but that is robust and flexible enough to be easily adapted to varied engines and operating conditions.

The deterministic nature of the cycle-to-cycle variations in heat release that are encountered implies that control should be possible. Conceivably, a more physically informative model could be used for control without this extreme level of tuning. However, traditional model-based control using models that are more detailed is infeasible because a model that would incorporate all of the relevant physics cannot run in the time available for calculations between engine cycles. Indeed, even the simple model described in Chapter 5 requires time on the order of a few seconds to simulate a single engine cycle on modern PC hardware. Because of this, a "black-box" method of control, such as a neural-network controller, which is able to control the system in the absence of detailed information about the internal states, is attractive.

Artificial intelligence approaches such as neural network based controllers seem ideally suited for such a problem. A controller that can learn the dynamics of the engine

without requiring substantial investment in offline tuning for each possible condition on each possible engine would be much more feasible to implement in production environments.

A variety of artificial intelligence applications to combustion control and modeling were reviewed in 2003 by Kalogirou (39). He reviews applications of neural networks, genetic algorithms, fuzzy logic, neuro-fuzzy logic, and other hybrid AI systems to combustion and internal combustion engines. A variety of different applications are discussed, including modeling for knock detection, building performance maps to relate engine power output or emissions to state variables, spark timing control, and various other applications. However, cycle-by-cycle control of combustion variations is not included in the 109 references he reviewed.

1.2.3.3 Description of Neural Network Control Scheme. More recently, artificial neural networks have been applied to the problem of cyclic variations in dilute SI engine operation by researchers at the Missouri University of Science and Technology (formerly named the University of Missouri – Rolla). He, et al. (40; 41) developed a backstepping NN control algorithm to control the in-cylinder equivalence ratio of a lean-burn SI engine. The architecture of this network was based on the model of Daw, et al. (26), and required information about the actual fuel and air masses in the cylinder at the beginning of an engine cycle, so could not be practically implemented in hardware, but was a first step in controlling the Daw model, which shares the same dynamics. For near-stoichiometric conditions, some models such as that of Tunestal (42) are able to determine in-cylinder AFR based on cylinder pressure data, but for lean equivalence ratios this information is not available.

The addition of another neural network to the system by Vance, et al. (43) allowed adaptation of the controller to use indicated combustion heat release as a feedback parameter. Singh, et al. (44), and Vance, et al. (45) further developed this control scheme to account for dilution through EGR rather than lean operation. Since these controllers are particularly of interest in later chapters, their application is described here.

The basic design of the neural-network controllers described in references (43; 44; 45; 46) is as follows. First, the crank-angle resolved in-cylinder pressure data are

integrated from spark to exhaust valve open (EVO) in a simple first law thermodynamic model, neglecting heat transfer and other losses, to obtain a value for the heat released in the combustion process, as described by Heywood (9).

While cylinder pressure sensors are not yet typically used in production engines, they are becoming robust enough for installation in such environments, as reported by Herden and Küsell (47), and are becoming more affordable. Fitzpatrick, et al. (48) described the design of an optical in-cylinder pressure sensor, consisting of a diaphragm, optical fiber, and integrated processing chip containing a light source, photodiode, and interferometer, and report good correspondence with measurements from reference sensors. Sellnau, et al. (49) additionally described an engine control system utilizing a low-cost pressure sensor mounted in the spark plug boss.

Mladek and Onder (48) presented a model for determining the inducted air mass from in-cylinder pressure measurements that would allow such sensors to replace the mass airflow sensor, allowing for cost-effective use of in-cylinder pressure sensors, which would provide the necessary feedback for this control scheme. With the growing likelihood that pressure measurement will become feasible, other control schemes that rely on cylinder pressure, such as that of Müller, et al. (51) are currently being developed by the automotive industry.

Given the availability of measured cylinder pressure, the heat release for a cycle, $y(k)$, is then numerically integrated from the measured in-cylinder pressure and known engine geometry using the trapezoid rule. A simple first-law thermodynamic analysis is used, neglecting heat transfer and other losses, as in Equation 1.1:

$$y(k) = \int_{SOC}^{EVO} \frac{\gamma}{\gamma-1} p(\theta) \frac{\partial V}{\partial \theta} + \frac{1}{\gamma-1} V(\theta) \frac{\partial p}{\partial \theta} d\theta \tag{1.1}$$

The cylinder volume, V, can be calculated from the crankshaft position and engine geometry parameters using Equation 1.2:

$$V(\theta) = V_c \left(1 + \tfrac{1}{2}(r-1)\left(R + 1 - \cos\theta - \sqrt{R^2 - \sin^2\theta}\right)\right) \tag{1.2}$$

This heat release value is fed as an input into a neural-network observer, which having been trained online determines an estimate of the heat release that will occur for the next engine cycle. Since neural networks are well suited to learning patterns, an effective observer should be able to detect the recurring patterns in the sequence of heat release events that is illuminated by the symbol sequence analysis described previously.

The output of this observer is then passed on to a neural-network controller, which also has information available regarding the previous cycle's heat release and the control input that was given for that cycle. This neural-network controller is also designed to learn online; during some initial learning period, it will train itself as to how changes in the control input cause changes in the measured output (combustion heat release). This controller will determine the desired perturbation to the control input for the next cycle, and this new control input will be used for the next engine cycle. Figure 1.1 shows the relationship between observer and controller neural networks and the engine.

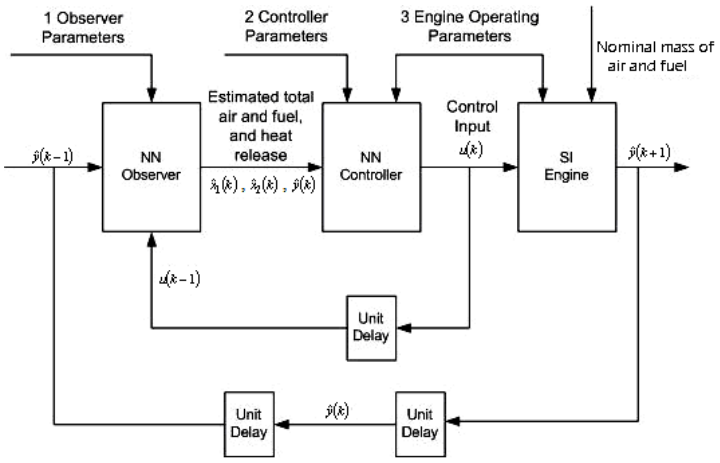

Figure 1.1. Controller block diagram, from Vance, et al. (46).

The controller makes use of two neural networks. The first generates a "virtual control input" (desired fuel and air masses for the next cycle) based on the heat release feedback, desired heat release, and observer estimates. The second takes this "virtual control input" along with the observer estimates, and determines how to perturb the fuel to drive the engine to the desired fixed point. This structure is a remnant of an earlier control concept of controlling equivalence ratio, with the first NN added as a patch onto the other controller NN in order to allow control targeting heat release rather than equivalence ratio.

It is apparent from the brief description above that a short time is available for all calculations to be completed between the end of combustion in one cycle and the start of fuel injection for the following cycle. Figure 1.2 illustrates these timing considerations.

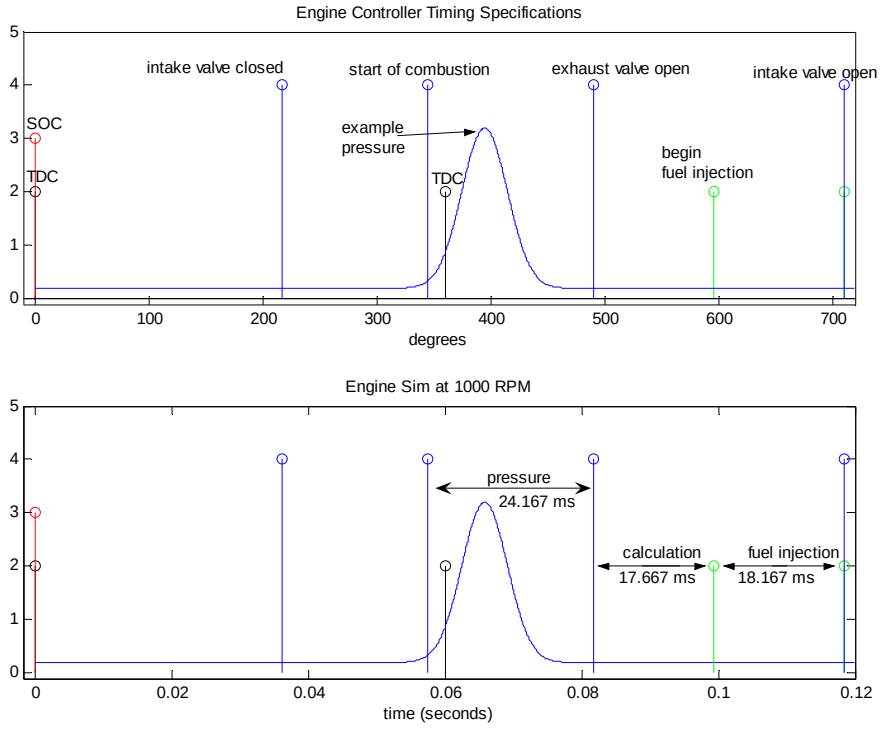

Figure 1.2. Timing considerations at 1000 RPM, from Vance, et al. (46)

The current controller is implemented in software on dedicated PC-based hardware, running a Pentium III 800 MHz CPU. Actual total calculation times, including the integration to determine heat release and all observer and controller calculations, take on the order of 3-4 ms on this hardware, which is well within the 17-18 ms window between the end of combustion and the beginning of the time allotted for fuel injection. Mathematical details and equations for both the neural network observer and controller can be found in references (45; 46).

1.2.4. Engine Modeling. While much prior work exists in engine modeling, most of it falls into one of two categories. Those in the first group are focused on obtaining an intensely detailed simulation of a single representative engine cycle and are computationally intensive. Secondly, other simple mathematical models have been developed to describe the dynamics of cycle-to-cycle variations, but these lack the physical insight given by the more complex, detailed, single-cycle models and do not contain sufficient detail to be truly predictive of individual cycle behavior, but must instead be tuned to simulate experimental data; they do, however, give useful insight into observed trends.

1.2.4.1 Physically Detailed Models. A great deal of effort has been focused on accurately simulating individual cycles in internal combustion engines. The definitive code in this field is KIVA (52), which was developed at Los Alamos National Laboratory for CRAY supercomputers, and is freely available (53). Further development produced KIVA-II (54) and KIVA-3 (55). In their current state, the KIVA-based codes combine multi-dimensional computational fluid dynamics and fuel spray dynamics with chemical kinetics calculations to model combustion and the formation of pollutants and with detailed heat transfer simulations. Submodels and derivatives such as KIVA3V-LITE (56) have also been developed by other research groups such as the Engine Research Center at the University of Wisconsin-Madison. These models are very detailed, but may take upwards of a full day to simulate a single engine cycle on a CRAY supercomputer. Implementation on massively parallel hardware at Oak Ridge National Laboratory has allowed for faster simulations (57), but the complexity is still far too high to make the simulation of thousands of consecutive cycles feasible.

Another, less involved, approach to engine combustion modeling is to neglect the multidimensional CFD solutions of the Navier-Stokes equations that are so computationally intensive in the KIVA model, while retaining thermodynamic information and other physics. Such zero-dimensional models may be as simplistic as assuming that the entire cylinder is homogeneous, or may consider multiple "zones," each of which is internally homogeneous. Two-zone models, for instance, track the burned and unburned gases separately, while remaining zero-dimensional with regards to resolution of any fluid dynamics.

Much of the recent work with this type of model has been done by Caton (58; 59). In his work, the thermodynamic model is coupled with a Wiebe function to specify the relation between mass fraction of burned gas and the crank angle (or time). The global thermodynamic constraints of the model described in Chapter 5 are based upon Caton's work, wherein these thermodynamic constraints are combined with a turbulent combustion model (60) in place of the Wiebe function.

The first turbulent entrainment combustion model for SI engines was developed by Blizard and Keck (61), and is based on mixing length theory with parameters of turbulent entrainment speed and a characteristic eddy radius. This model was experimentally validated on a single-cylinder SI engine for equivalence ratios from 0.7 – 1.5. Tabaczynski, et al. (62) developed an improved model, which more accurately predicts the variation in combustion duration with equivalence ratio, emphasizing the role of the Taylor microscale. This model was later refined by Tabaczynski, et al. (63) to further account for the development of the flame and changing length scales of the turbulent eddies. This form of turbulent entrainment model is described by Stone (64) and by Heywood (9) in their textbooks on internal combustion engines, and is used in the model described in Chapter 5.

Other methods of accounting for the effect of turbulence on the burning rate have also been used in SI engine combustion modeling. Keck (65), for example, used a wrinkled laminar flame approach to correlate burning rates to engine geometry and operating parameters, in effect enhancing the flame area through empirical correlations rather than considering details of the turbulence involved. De Petris, et al. (66), following up on the work of Gouldin, et al. (67), use fractal flame models to account for

the effects of turbulent wrinkling on the burning rate. This approach requires model constants and is more well-suited to three-dimensional simulations. A Covarial analysis approach has also been shown to better predict the effects of the turbulence level in multidimensional simulations for lean conditions by Pajot, et al. (68). The approach of Tabaczynski, et al. (63), however, is applicable to zero-dimensional models and offers more insight into the combustion physics without prohibitive computational overhead.

Metghalchi and Keck (69) developed improved laminar flame speed correlations for isooctane, methane, and indolene. These correlations account for the effects of temperature, pressure, and composition on the laminar flame speed. These correlations are described by Turns (70), Stone (64), and Heywood (9). Correlations for other fuels, including ethanol, propane, and alcohol/water blends were developed by Gülder (71), and burning velocities of ethanol-isooctane blends (E-10 and E-20) were reported by Gülder (72). These additional flame speed data for various fuels can be inserted into the laminar flame speed correlations contained within the turbulent burning models described above.

The initial work in integrating these thermodynamic models to simulate the nonlinear dynamics involved in highly dilute SI engine operation is described in Chakravarty, et al. (60), using a fixed multiplicative factor to account for the enhancement in burning rate due to turbulence. Further development of the model to include the turbulent entrainment approach of Tabaczynski, et al. (63) is described in Chapter 5. The Chakravarty, et al. model was later coupled with the Ricardo WAVE engine simulation package by Edwards, et al. (73), allowing for simulation of the effects of many external engine parameters.

1.2.4.2 Nonlinear Dynamics Models. Another approach to modeling the cyclic variations of interest is to empirically create a mathematical model that will simulate the time series of combustion events. Fitted map functions such as those described by Wagner, et al. (38) exemplify this approach. There is no explicit physical basis for the function chosen, nor can any physical insight gained by examining internal state variables since none exist, but it can predict a combustion event based on the prior events with some accuracy.

It is also possible to combine nonlinear dynamics insight with some limited physical details. This approach is used by the model of Daw, et al. (26), but with the

addition of some simple physics. This model includes conservation of mass combined with an empirical mathematical function, which is used to model the nonlinear dynamical relationship between the charge composition and combustion heat release.

The Daw, et al. (26) model is described here in detail, as it is the primary existing nonlinear dynamical model of the cyclic variations encountered in lean SI engine operation and has relevance to control applications. The composition of the cylinder charge is tracked, with some level of residual gases from each cycle being carried over to the next. An empirical, non-linear function is used to relate the charge composition to the completeness of combustion, and thus to an output heat release.

The air available for combustion in the present cycle is given by Equation 1.3:

$$m_a(k+1) = m_{a,new}(k+1) + F_{res}(k)m_a(k) - \left(\frac{A}{F}\right)_S F_{res}(k)CE(k)m_f(k) \tag{1.3}$$

The total mass of fuel present is given by Equation 1.4:

$$m_f(k+1) = m_{f,new}(k+1) + (1 - CE(k))F_{res}(k)m_f(k) \tag{1.4}$$

Combustion efficiency is a nonlinear function of the equivalence ratio, and is an empirically chosen sigmoid function that is tuned to simulate engine behavior. Equation 1.5 shows this function:

$$CE(k) = \frac{1}{1+100^{\frac{-(\varphi(k)-\varphi_m)}{\varphi_u-\varphi_l}}} \tag{1.5}$$

$$\varphi_m = \frac{\varphi_u-\varphi_l}{2} \tag{1.6}$$

φ_u and φ_l are parameters chosen to tune the equation to fit observed engine behavior, and $\varphi(k)$ is the equivalence ratio at the given cycle, defined as the actual fuel/air mass ratio divided by the stoichiometric fuel/air mass ratio:

$$\varphi \equiv \frac{\left(\frac{F}{A}\right)_A}{\left(\frac{F}{A}\right)_S} \tag{1.7}$$

The model output, heat release, shown in Equation 1.8, is assumed to be proportional to the mass of fuel burned:

$$y(k) = \alpha m_f(k) CE(k) \tag{1.8}$$

α is a constant of proportionality chosen to scale the model results to match actual engine combustion heat release.

Stochastic perturbations can be imposed on input parameters (φ, etc.) in order to account for the non-deterministic or higher order variations encountered. The residual fraction, equivalence ratio, Gaussian noise level, φ_u, and φ_l are specified as inputs. The masses of fuel and air are calculated based on the input equivalence ratio, the residual fraction, and an assumed (constant) total charge mass. The composition of the cylinder gases at the end of combustion (amount of fuel and air remaining unburned) is carried over to influence the next cycle, as seen in the m_f and m_a equations above. Each engine cycle is a discrete-time event; there is no detail included of the progression of combustion through the cycle. A modified version of the model that also accounts for EGR (inert dilution) additionally tracks what fraction of the residual gases is inert, in addition to fuel and air, through basic stoichiometric relations. The level of dilution is used to modify the equivalence ratio that is used as an input to Equation 1.5.

2. EXPERIMENTAL SETUP AND FACILITIES

2.1. INTERNAL COMBUSTION ENGINE LABORATORY FACILITIES

Experimental data were collected on single-cylinder research engines in the Internal Combustion Laboratory. Descriptions of each experimental setup and of the instrumentation used follows.

2.1.1. Ricardo Engine Setup. The primary engine used in experiments is a single-cylinder Ricardo Hydra research engine, with a modified Ford Zetec cylinder head having the same geometry as a 2.0L 4-cylinder Zetec engine. The engine geometry is listed in Table 2.1. This engine is mounted to an electric dynamometer with speed control to maintain a fixed engine speed, but no other feedback control.

Table 2.1. Ricardo engine specifications

Bore	84.84 mm
Stroke	88.00 mm
Displacement volume	497.4 cm^3
Compression ratio	9:1
Intake valve open (IVO)	5° BTDC @ 0.15 mm lift
Intake valve close (IVC)	47° ABDC @ 0.15 mm lift
Exhaust valve open (EVO)	48° BBDC @ 0.15 mm lift
Exhaust valve close (EVC)	4° ATDC @ 0.15 mm lift

Figure 2.1 shows this engine, as configured. A 1°-resolution crankshaft encoder is installed on the front of the crankshaft, to provide timing signals for both the data acquisition system and the fuel injection and ignition drivers. The dynamometer can be seen towards the left of the picture.

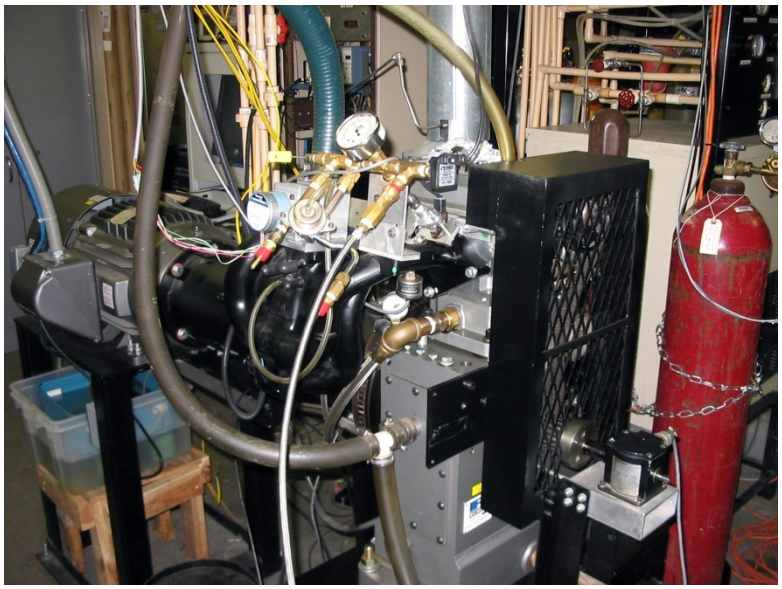

Figure 2.1. Ricardo engine setup

Fuel injection is controlled by a fixed timing signal generated by a Berkeley Nucleonics Corporation model 500 delay generator, and then fed through a Ford fuel injector driver that has been modified to take an external timing input. Fuel pressure is supplied by a pressurized tank, with pressure maintained by regulated dry compressed air, to eliminate the fluctuations that would be encountered with an electric fuel pump.

The BNC-500 delay generator is also used to generate a constant spark timing signal, which is sent to an ignition module, which drives the coil. A production Ford coil-on-plug ignition coil and spark plug are used.

Load is controlled by a throttle plate, via an affixed 1.8° stepper motor. There is no feedback load control on the dynamometer system; a fixed throttle position is set for an operating condition and maintained throughout a test. The control system is not designed for transient testing.

More details on the engine setup, particularly concerning modification of the cylinder head and other components from the stock four-cylinder to a single-cylinder configuration, and related to the installation and maintenance of the engine, can be found in Evers (74).

Inert dilution is provided by bottled nitrogen gas, which is introduced into a surge tank far upstream of the throttle, to allow for complete mixing with the intake air. The nitrogen flow rate is controlled by manual adjustment of a needle valve.

2.1.2. CFR Engine Setup. A single-cylinder CFR research engine was also available, and was used for some early tests of the controller. This is the same engine that was used by Wagner (75) for characterization of the dynamics of cyclic variability in lean SI engine operation. Figure 2.2 shows this engine, as installed. The red and white striped box is a shielded enclosure around the ignition coil, to reduce spark noise.

Figure 2.2. CFR engine setup

Fuel, spark, and throttle control are largely the same as for the Ricardo engine, with fixed timing signals supplied by a delay generator taking a signal from the shaft encoder, and a stepper motor for throttle control. Cooling and intake systems are shared with the Ricardo engine setup. The engine geography specifications are shown in Table

2.2. Like the Ricardo engine, the CFR is mounted to an electric dynamometer with speed control but no other feedback control systems in place.

Table 2.2. CFR engine specifications

Bore	84.84 mm
Stroke	88.00 mm
Displacement volume	497.4 cm^3
Compression ratio	9:1
Intake valve open	5° BTDC @ 0.15
Intake valve close	47° ABDC @ 0.15
Exhaust valve open	48° BBDC @ 0.15
Exhaust valve close	4° ATDC @ 0.15

2.1.3. Data Acquisition and Instrumentation. PC-based data acquisition systems were used to acquire crank angle resolved pressure data for heat release calculations. The data acquisition systems used both employed National Instruments PCI data acquisition cards: the primary system has a model PCI-6071E, and a secondary system that is also utilized has a model PCI-MIO-16E-4. Both systems receive a top dead center (TDC) signal and a crank angle signal from an optical crankshaft encoder on the engine. The TDC signal is used to trigger the start of data acquisition, indexing the beginning of each data file to the start of an engine cycle. The crank angle signal is used as a clock to trigger acquisition of each sample.

This crank-angle trigger is conditioned using a Stanford Research model DG535 delay generator, in order to eliminate errant triggering of the data acquisition systems due to induced spark noise. The shaft encoder on the Ricardo engine is a Datametrics model BM-360-5SE-1, and that on the CFR is an identical Lucas Ledex HD30 DM-360-5SE-1.

Both engines are fitted with in-cylinder Kistler 6061A water-cooled pressure transducers. The signal from the pressure transducer is processed by a Kistler Type 5010

charge amplifier, to produce a voltage signal, which is read by the data acquisition system.

In addition to the crank-angle resolved cylinder pressure data, many other parameters are recorded on a longer time-scale and averaged over several minutes, to obtain a mean value for the entire period of 1000 or more engine cycles over which data is acquired. The primary data acquisition system was used to acquire the crank-angle resolved data, while the secondary system simultaneously records these other data.

The intake air flow rate is measured by means of a Meriam 50MW20-1-1/2 laminar flow element (LFE) upstream of the surge tank. The pressure drop across the LFE is measured by a GE Druck LP 1000 differential pressure transducer. Absolute pressure at the inlet of the LFE is measured by an Omega PX209-015A5V absolute pressure transducer. Intake air temperature and humidity are measured in the surge tank by an Omega HX96-3-V-D. These signals are sampled by the secondary data acquisition system at 100 Hz over several minutes and averaged to determine the mass air flow rate into the engine.

Fuel pressure is measured at the injector manifold by an Omega PX161-060G5V pressure transducer. This signal is also sampled by the secondary data acquisition system, and along with the fuel temperature and injector pulse width, is used to calculate the mass of fuel injected per engine cycle.

Mass flow rate of nitrogen used for dilution is measured using a Sensirion CMOSens Type EM1_V4R0V_1A digital mass flow sensor, which is read by the primary data acquisition system via a serial (RS-232) interface.

Both engine setups also have NTK TL-6111 UEGO sensors mounted in the exhaust manifolds for measurement of equivalence ratio. The voltage output of the UEGO sensor is measured and averaged with a Fluke 83 multimeter.

Temperatures are measured at a number of locations by type K thermocouples, and read individually by a digital thermocouple reader. Locations of thermocouples for both engine setups are listed in Table 2.3.

Table 2.3. Thermocouple locations

TC #	Ricardo	CFR
1	Oil cooler	HX cooling loop supply
2	Block oil	HX cooling loop return
3	Engine coolant supply	HX engine loop supply
4	Engine coolant return	HX engine loop return
5	Head coolant return	Cooling tower supply
6	Cylinder coolant return	Pressure transducer cooling water
7	Exhaust	Exhaust
8	Pressure transducer cooling water	Head coolant
9	Intake air stream at LFE	Intake air stream at LFE
10	Fuel	Block oil
11		Fuel

3. FILTERING DETERMINISTIC EVENTS FROM EXPERIMENTAL DATA

3.1. NONLINEAR DYNAMICAL DATA ANALYSIS

In order to better understand the role that highly dilute SI combustion could play if controllers are able to effectively reduce the problematic cyclic variations encountered in this mode of operation, experimental engine data were analyzed using techniques of nonlinear dynamics. The engine cycles that correspond to deterministic sequences of events are detected and removed, and the remaining cycles, which should vary stochastically, are examined. In this manner, the improvement that could be hoped for with effective control can be estimated.

3.1.1. Symbol Sequence Analysis. Nonlinear dynamics and chaos theory have spawned a number of new mathematical analysis techniques for time-series data. One such useful method for analysis is to partition the data, representing each partition with a symbol, and then consider patterns in the sequence of the symbols that represent these partitions. This technique, known as symbol sequence analysis, gives substantial insight into the behavior observed in highly dilute SI engine operation. Finney, et al. (36) have previously documented this technique, but an explanation is repeated here.

When using symbol sequence analysis, data are first divided into equiprobable partitions so that there are an equal number of data points in each partition, each of which is represented by a symbol. The simplest example of this is binary partitions, where values below the median are represented by "0" and values above the median are represented by "1". Figure 3.1 illustrates a sequence of 100 heat release cycles, showing binary partitions.

It is, of course, also possible to use more than two partitions; a different number system would result. For example, eight partitions would be represented by the octal numbering system, with symbols "0," "1," "2," "3," "4," "5," "6," and "7." With all of the data thus partitioned, sequences of these symbols can be considered. Sequences that occur more often than would be expected from stochastic variations must be deterministic. On a histogram plot, these sequences will appear as peaks rising above the background "noise."

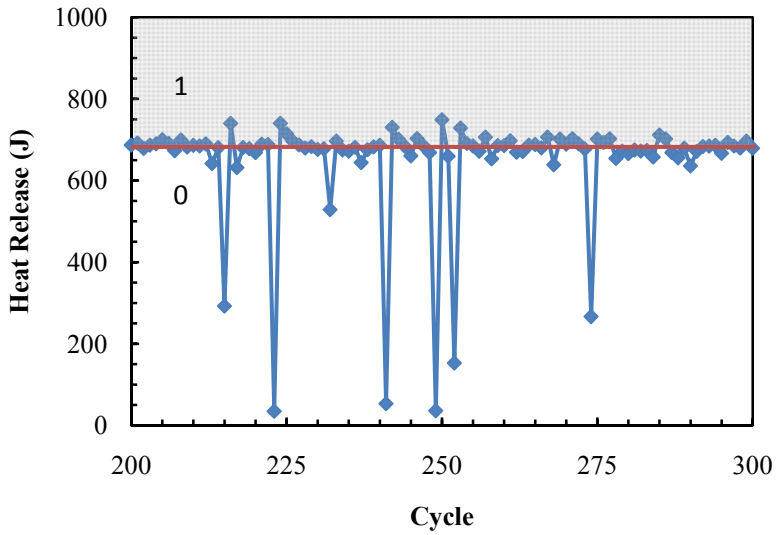

Figure 3.1. Heat release sequence for dilute operation, illustrating symbolic partitions

An example symbol sequence histogram for a set of engine heat release data at a high dilution level for the Ricardo test engine is shown in Figure 3.2, for binary partitions and a sequence length of six.

For purely random data, all sequences would occur with the same frequency, due to the partitions being equiprobable. The frequency that would be observed for all sequences for purely random data is given by Equation 3.1.

$$F_b = \left(\frac{1}{n_{part}}\right)^{l_{seq}} \tag{3.1}$$

This baseline frequency is shown by the red line on Figure 3.2. In this case, several peaks rise above this baseline, but two are particularly noticeable. The decimal numbers 21 and 42 convert into 010101 and 101010 in binary, representing sequences 0-1-0-1-0-1 and 1-0-1-0-1-0, or alternating low-energy and high-energy combustion events.

An example of a symbol sequence histogram for eight partitions, but a sequence length of only two, is shown in Figure 3.3.

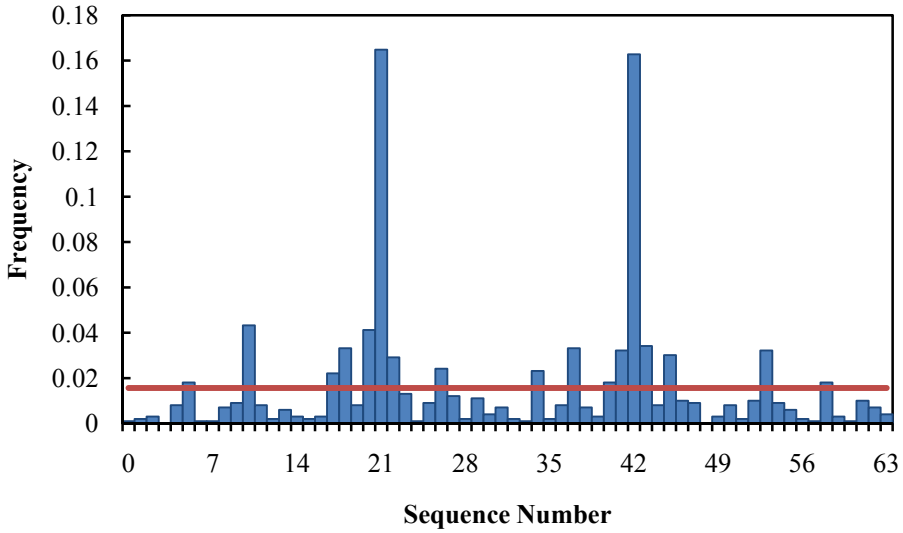

Figure 3.2. Symbol sequence histogram for 2 partitions, sequence length 6

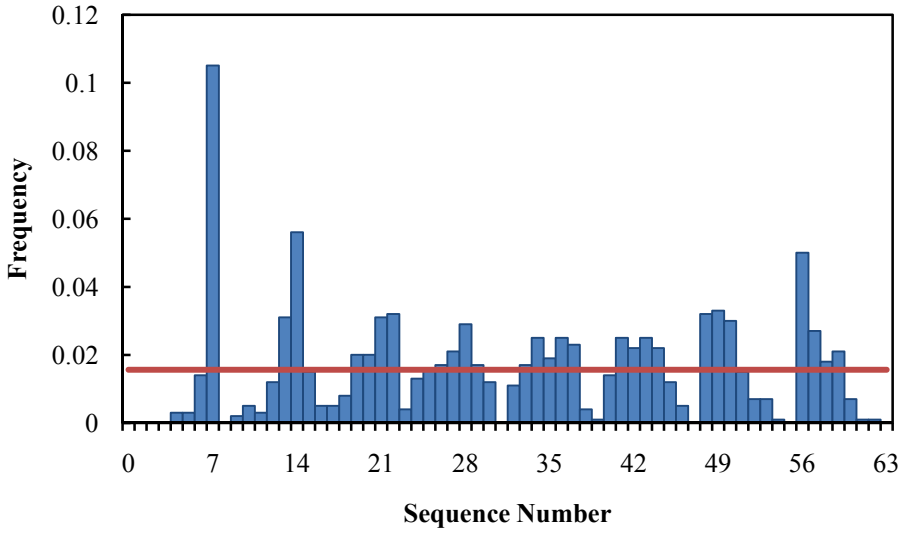

Figure 3.3. Symbol sequence histogram for 8 partitions, sequence length 2

In this case, the low to high and high to low energy combustion events are also dominant, with sequence number 7, which equates to the symbol sequence 0-7, being the most frequent pattern encountered. This indicates that a combustion event in the lowest energy partition is often followed by a combustion event in the highest energy partition. Also, it is seen that events in the highest partition, 7, rarely occur after other high energy events: valleys at 47, 55, and 63 correspond, when converted from decimal to octal numbers, to sequences 5-7, 6-7, and 7-7, so occurrences of the highest energy cycles almost never follow a prior cycle that also exhibited a high energy combustion event. Similarly, low energy events are typically followed by high energy events rather than by other low energy cycles: an event in the lowest partition, 0, are nearly always followed by events in one of the two highest partitions, 6 or 7, and never by events in other low energy partitions, 0 through 3.

Mathematically, it would be valid to combine any number of partitions with any sequence length. However, in order for the results to be statistically meaningful, the size of the data set must be sufficiently large to outnumber significantly the possible symbol sequences being considered. For both examples shown above, there are 64 possible sequences of symbolized data, so that an equal amount of data is considered. The 1000 engine cycles worth of data that are acquired for each operating condition in this work are more than sufficient for this analysis to be valid with the partitions and sequence lengths presented. If, however, one were to consider for example eight partitions, but with a sequence length of six, that would give 8^6, or 262,144 possible symbol sequences. In order to maintain the same ratio of data to possible sequences as in the earlier cases, nearly 4,000,000 engine cycles would be required for each operating condition, which corresponds to nearly six days of continuous 1000 RPM operation. Thus, overly long sequences combined with a large number of partitions would require impractically large data sets.

Additionally, while the use of a greater number of partitions offers higher resolution, as the number of partitions is increased, the distinction of sequences that was facilitated by the partitioning of the data becomes more difficult. At the limit, a sufficiently large number of partitions would be just the same as the raw, unpartitioned

data. There is therefore an advantage in maintaining few enough partitions to distinguish the sequences that are present.

3.1.2. Shannon Entropy. In order to choose an appropriate sequence length, it would be informative to consider whether there is a factor inherent in the engine behavior itself that warrants a particular choice. A modified form of Shannon entropy can be used to quantify the deviation of heat release sequences from randomness (75; 76). This modified Shannon entropy is defined by Equation 3.2.

$$H_s = \frac{1}{\log n_{seq}} \sum_k p_k \log p_k \tag{3.2}$$

A Shannon entropy of one indicates random data, while values less than one indicate correlation between sequential data points. The Shannon entropy can be used to determine the optimal sequence length to consider in a symbol sequence histogram, since the sequence length giving the lowest Shannon entropy indicates the greatest presence of determinism in the data. Figure 3.4 illustrates the variation of Shannon entropy with sequence length for selected sets of lean SI engine data. Dilute stoichiometric (high EGR) operation yields essentially identical results (76). Table 3.1 shows the change in Shannon entropy with increasing inert dilution, for both the 8-2 and 2-6 analyses. The decrease in Shannon entropy indicates that a controller should have a greater effect at these higher dilution levels than at lower levels, where the dispersion encountered has a more random nature, as expected.

The minimum in Shannon entropy around the sequence length of six cycles indicates that the influence of previous cycles on the heat release extends back to six cycles in the past. This is in agreement with earlier results for both lean and high EGR engine operation (75; 76), and indicates that approximately six cycles are required for the residual gas to be fully scavenged so that the event of an earlier cycle will have no significant effect. Therefore, it appears that a sequence length of six would be a good choice for evaluating such engine data.

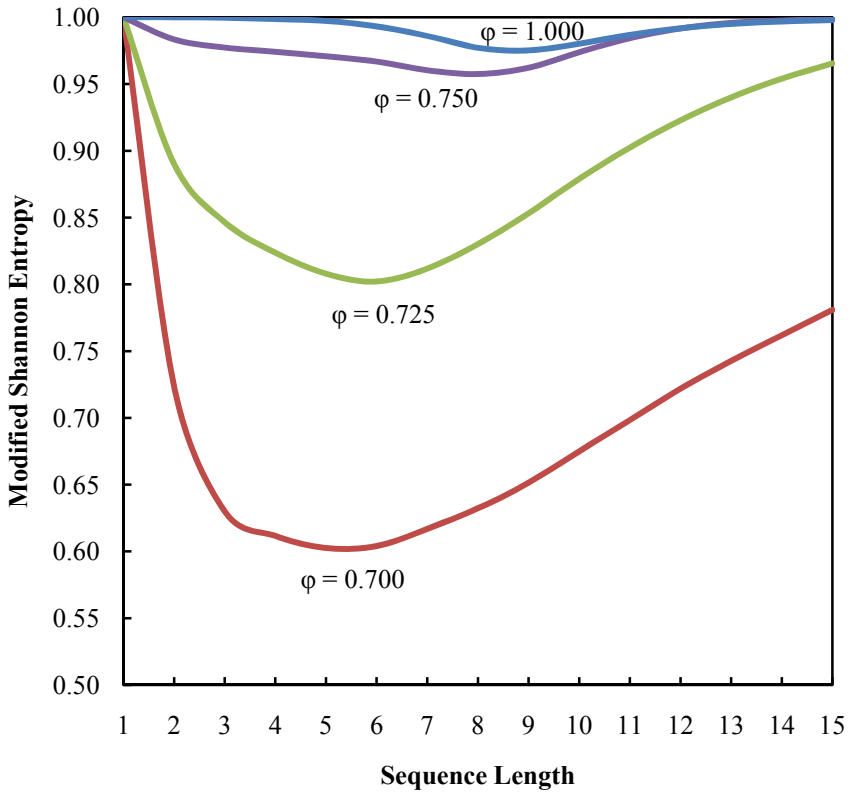

Figure 3.4. Plot of modified Shannon entropy for engine data at several dilution levels

Table 3.1. Shannon entropy variation with diluent level

Diluent %	H_s 2-6	H_s 8-2
0	0.992	0.991
20	0.988	0.993
24	0.929	0.943

Another consideration that would be appropriate if applying this analysis method is the number of cycles about which a controller has information. The current generation of controllers being developed only retains the prior cycle heat release in memory, and does not consider earlier cycles (43; 44; 45; 46). So, for a realistic evaluation of what

these controllers are doing, a sequence length of only two could be more appropriate, since that is as much information as the controller has available to it.

Due to these considerations, two sets of parameters were used for data analysis. For evaluation of the current controllers that only have memory of the previous cycle, 8 partitions are used, considering a sequence length of 2 cycles (8-2). The larger number of partitions more closely represents the higher resolution of the actual controller, compared to the binary partitions, with values of 0 and 1 only. To project what could be obtained by eliminating longer patterns, 2 partitions are used, considering sequences with a length of 6 cycles (2-6). For these longer sequences, there was not enough data acquired to use octal partitions, so binary partitions were used, maintaining the same amount of data and number of possible sequences.

3.2. APPLICATION TO ESTIMATION OF IMPROVEMENT IN EFFICIENCY OF CONTROLLED OPERATION

The data analysis techniques described in the previous section will be applied here to the question of what improvements a controller could be expected to effect on the behavior of dilute SI engine combustion. The controllers described earlier, such as those developed by Singh, et al. (44) and Vance, et al. (45; 46) offer a promising approach to addressing the cyclic variability encountered in dilute SI engine operation. This analysis will show how much improvement could be gained if such a controller is fully effective.

The aforementioned symbol sequence analysis illustrates which sequences of heat release events recur often enough to be attributed to deterministic, rather than stochastic, causes. It should therefore be possible to keep track of which engine cycles contribute to those sequences that repeat too often to be random. If these cycles are eliminated from the data set, and only the remaining "cleaned" cycles are considered, then what remain are only those cycles that deviate from the norm because of uncontrollable, random variations.

It is unreasonable to expect that any artificial control system would be absolutely perfect in detecting and eliminating all possible variations, so rather than all cycles that result in a peak in the symbol sequence histogram rising above the random baseline frequency being eliminated, some multiplier should be applied to the baseline frequency to determine the cutoff frequency to be used. So, if a controller could be expected to eliminate those variations that fall more than 20% outside the range accounted for by

stochastic effects, then all cycles contributing to peaks that rise higher than 1.2 times F_b would be removed. The desired level of performance can be adjusted, but in order for a controller to be considered effective, it should at least be capable of pulling the variations to within 20% of the desired value; this value was used as a baseline in the analysis presented here.

The "cleaned" data that result after these deterministic cycles are removed are then used to calculate metrics of interest, such as the coefficient of variation (COV) of heat release, the net IMEP, and the fuel conversion efficiency. The COV of a parameter is defined as its standard deviation divided by its mean value. Only metrics that are based on cycle-resolved data can be analyzed by this method; changes in time-averaged values such as exhaust emissions measured by emissions benches with a significant delay time rather than fast-response sensors cannot be estimated, since individual cycle results must be removed from the body of data.

Caution should also be used in evaluating the dynamics of the "cleaned" data. Since many of the original cycles have been removed, gaps are present in the sequence. Symbol sequence analysis of these data sets will be deceptive, since events that did not actually follow one another would be paired up as if they had. It is therefore not generally valid to produce symbol sequence histograms for these modified data sets.

3.3. RESULTS AND DISCUSSION

3.3.1. Return Maps of Cleaned Data. While symbol sequence statistics cannot be used to evaluate data sets that are missing many points and that would improperly pair unrelated cycles in sequence, other methods can be used to illustrate that the remaining data are primarily those cycles that did not contribute to deterministic variations. One useful way of viewing this data is through return map plots. In these plots, the heat release for a given engine cycle is plotted against the heat release from the previous cycle. Purely stochastic, Gaussian variations will be a tight circle on the 45° diagonal, as in Figure 3.5, while other patterns indicate deterministic structure.

Figure 3.6, for example, shows the return map of heat release data acquired during lean operation on the Ricardo engine. The operating conditions were an equivalence ratio of 0.725, and a fixed base spark timing of 15° BTDC; the COV of heat release for these data is 25.47%.

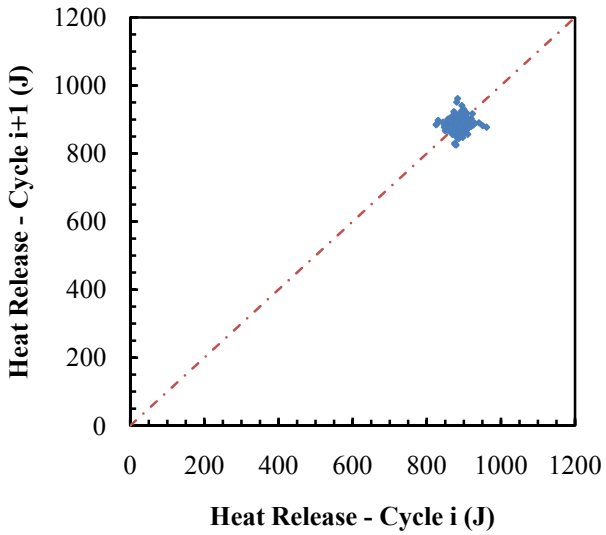

Figure 3.5. Heat release return map for Ricardo engine data, φ = 1

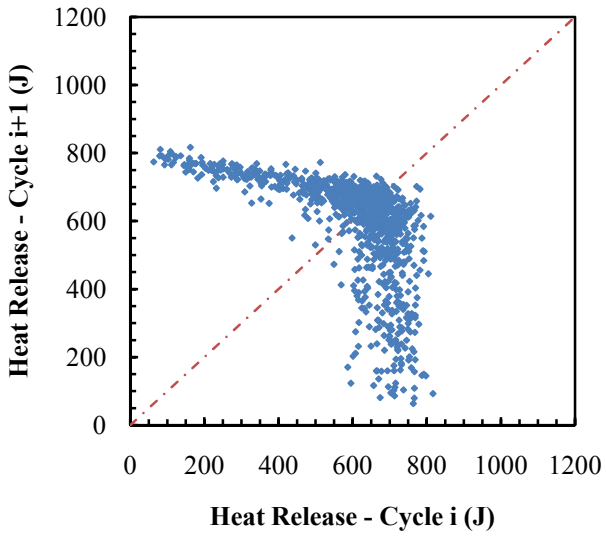

Figure 3.6. Heat release return map for Ricardo engine data, φ = 0.725

When symbol sequence analysis of these data is performed using 8 partitions and a sequence length of 2, and the cycles contributing to sequences that occur more than 1.2 times the baseline frequency are removed, the COV is reduced to 11.51%.

When only those heat release cycles that are immediately followed by another cycle that has not been removed from the data set are plotted on a return map, the dynamics of the remaining cleaned data can be observed, as seen in Figure 3.7. While a few outliers still remain, clearly the data are much more focused near the desired fixed point on the diagonal, with primarily stochastic variations.

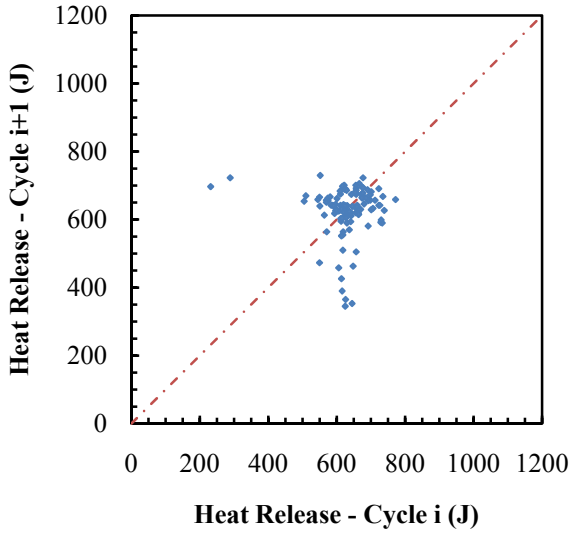

Figure 3.7. Heat release return map for cleaned Ricardo engine data, φ = 0.725

3.3.2. Comparison of Projection to Controlled Engine Behavior. In order to further illustrate the validity of this method of estimating controller performance, an example case of uncontrolled and controlled engine behavior was examined at an inert dilution level of 15%, which is high enough that some dispersion can be detected, and low enough that the controller being tested was relatively effective at reducing the cyclic

variations. Spark timing was held fixed at MBT for the uncontrolled case. This controller is capable of considering only the previous cycle, so a sequence length of 2 in the analysis is appropriate for comparison. Table 3.2 shows the COV of heat release for the uncontrolled case, the controlled case, and the projected improvements using both 8 partition/sequence length 2, and 2 partition/sequence length 6 analyses. The 8-2 projection is somewhat better than the actual performance of the controller, indicating that some further improvement is possible. The added improvement seen with the 2-6 projection implies that a controller designed to consider more prior cycles could improve upon the performance of a controller that considers only the immediately previous cycle. Elimination of longer patterns in the data yields a greater improvement than only removing recurring sequences of two consecutive cycles.

Table 3.2. Comparison of estimate to actual controller performance

	Dilution	COV HR	% Reduction in COV
Uncontrolled	15.3%	13.11%	
Controlled	15.2%	7.73%	41.04%
Projected 8-2	15.3%	4.71%	64.07%
Projected 2-6	15.3%	4.11%	68.65%

Examination of return maps shows the same. Uncontrolled operation in this case has problematically high cyclic variability, including numerous misfires, as shown in Figure 3.8. The controller is able to reduce the number of misfires significantly, but some deterministic variations remain, as the return map shown in Figure 3.9 illustrates.

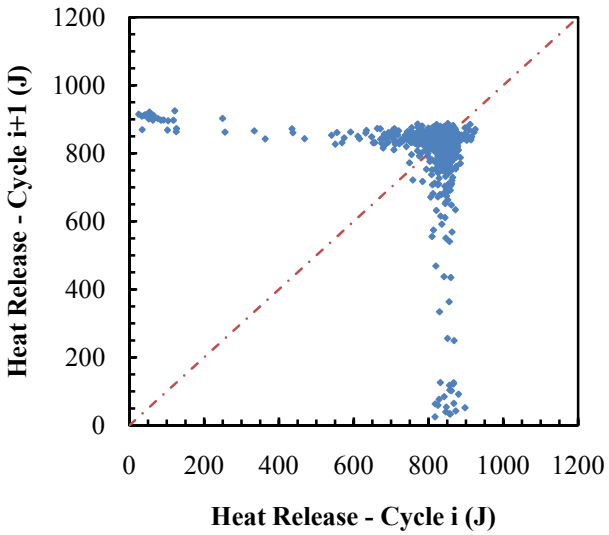

Figure 3.8. Heat release return map for 15% EGR, no control

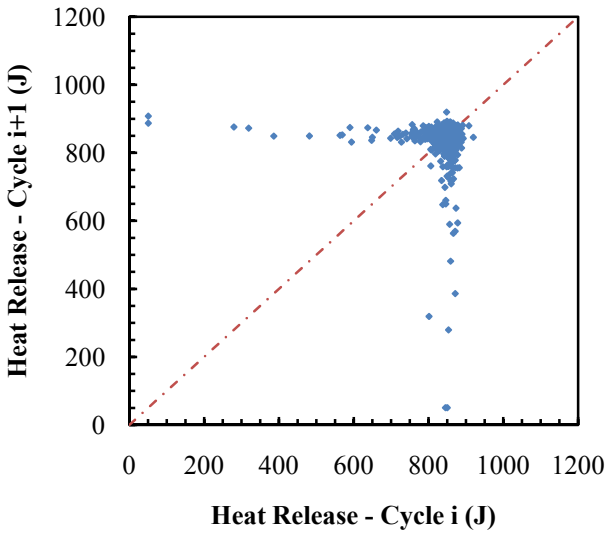

Figure 3.9. Heat release return map for 15% EGR, with control

Applying the symbol sequence analysis technique described above with the 8-2 parameters, all misfires are eliminated, and only a few outlying, apparently deterministic events remain. Figure 3.10 shows the return map for this case. The improvement is like that in the actual controlled case, but somewhat improved, indicating that even with information about only the prior cycle some further improvement in controller performance is possible.

Further, when longer sequences are considered, with the 2-6 parameters, Figure 3.11 shows an even greater reduction in variability, with nearly all cycles clustered in a stochastic pattern. This indicates that for this dilution level, a controller that has information about the events in more previous engine cycles offers slightly more potential than one considering only the immediately prior cycle.

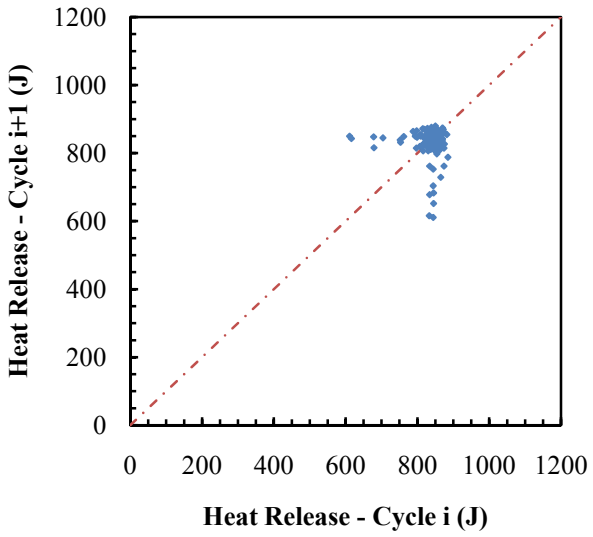

Figure 3.10. Heat release return map for 15% EGR, “cleaned” 8-2

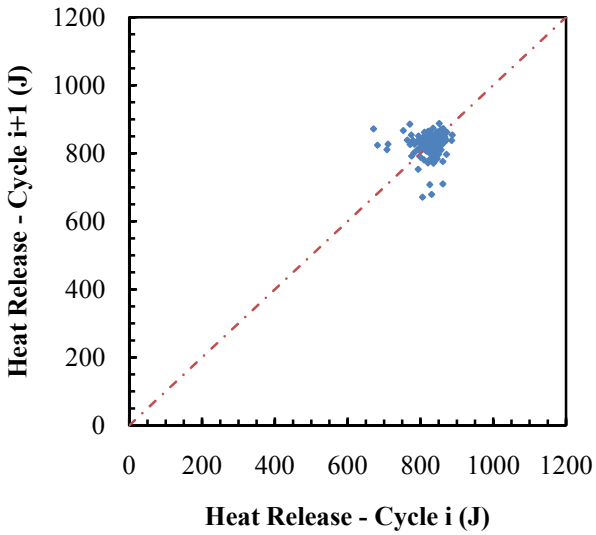

Figure 3.11. Heat release return map for 15% EGR, "cleaned" 2-6

While removing cycles from the data set precludes the use of sequential analysis for examining the improvement in a "cleaned" data set, it is possible to compare symbol sequence histograms for actual controlled operation to an uncontrolled reference. Controlled data acquired while operating at a higher dilution level, in this case 24% inert diluent, shows a significant (14.5%) reduction in the frequency of the most common deterministic peak, as seen in Figure 3.12. While this early generation controller is not yet able to prevent all of the deterministic events, the clear reduction illustrates the influence an effective controller will have on the dynamics of highly dilute engine operation. The Shannon entropy increased from 0.972 to 0.981 with control, further indicating a reduction in the deterministic variations. So it is clear from the multiple analysis techniques presented that the effect of this controller is to reduce deterministic variations in the engine output, and thus that examining data with the deterministic variations removed should give a reasonable estimate of the performance of a more effective controller.

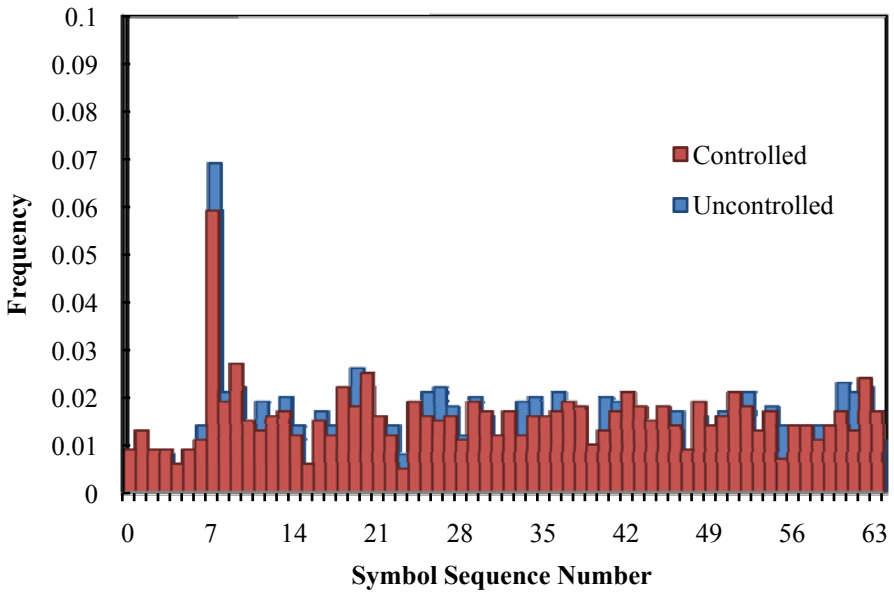

Figure 3.12. Reduction in deterministic variations with control

For this higher dilution level, the controller is not as effective at addressing the more severe variability encountered as it was for the lower dilution level, but some reduction in these problematic variations is observed, as shown above. The COV of heat release is shown along with the fuel conversion efficiency and the improvement in efficiency in Table 3.3.

Table 3.3. Controller performance for higher dilution level

	Dilution Level	COV HR	η_f	% Change in η_f
Uncontrolled	23.82%	28.93%	28.96%	
Controlled	23.90%	26.40%	29.20%	0.83%
Projected 8-2	23.82%	14.07%	30.60%	2.75%

In this case, there is substantial room for improvement even when only considering two cycles, as shown by the 8-2 projection. The fuel conversion efficiency, calculated as shown in Equation 3.3 plus appropriate unit conversion factors, is shown here as an example of an output parameter of interest that can be predicted by this method of projecting controller results. The average IMEP of the cleaned data is used in place of the actual average IMEP to calculate a cleaned fuel conversion efficiency for comparison. The engine speed, fuel flow rate, and lower heating value are fixed.

$$\eta_f = \frac{IMEP \cdot N}{\dot{m}_f \cdot LHV} \tag{3.3}$$

3.3.3. Dilute Operation for Load Modulation. A major advantage of the ability to utilize high levels of dilution is the potential to use dilution rather than throttling to control engine output. Data were collected at a constant engine speed for a range of load conditions using both throttling and dilution to reduce load. Spark timing was held at a constant value equivalent to MBT (minimum advance for maximum brake torque) for the case of no dilution. Figure 3.13 shows the indicated fuel conversion efficiency plotted against net IMEP for all of these data. It is apparent that as load is reduced, efficiency drops off slightly for throttled operation, while initially increasing significantly as dilution is increased. However, at high dilution levels, the fuel efficiency declines due to cyclic variability and reduced burn rates. The actual level of dilution at which this occurs will vary based on engine design parameters that affect dilution tolerance, but once variations become significant, they follow the same patterns for SI engines in general. For three of the dilute cases where cyclic variability was high enough to be problematic, the potential effectiveness of control on the engine behavior was estimated using 2-6 parameters, for a best-case scenario.

There is still an operational limit that is encountered, beyond which throttled, undiluted operation is more efficient than dilute operation due to the reduced burn rates at high dilution levels. However, the elimination of deterministic variations through control will stretch that dilution limit beyond where it is presently.

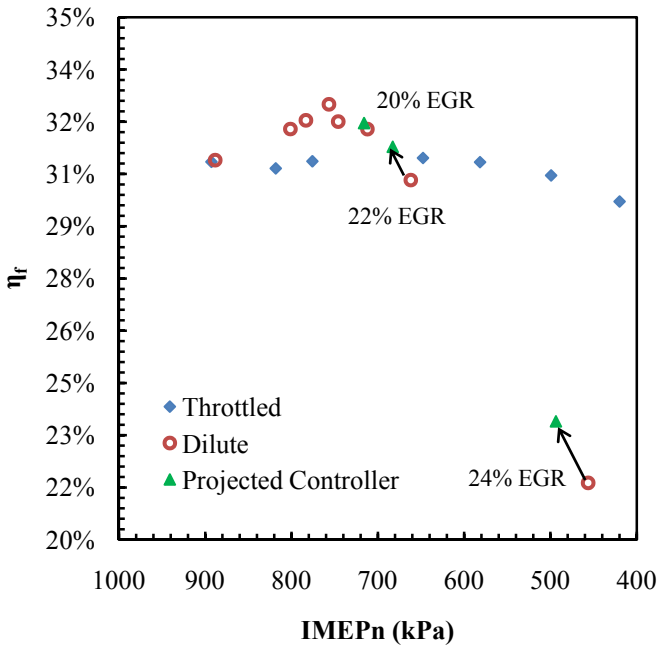

Figure 3.13. Projected efficiency gains with control for dilute SI operation

A controller that is able to meet the potential gains indicated by these estimates could extend the dilute SI operation envelope to the point where it can be used as an intermediate mode between traditional SI at very high loads and HCCI at low loads, if the HCCI/SI mode transition control problems are solved. Such a scenario is illustrated conceptually in Figure 3.14. Even if not used as an intermediary between SI and HCCI operation, dilute SI operation offers substantial efficiency gains and reduces the need for throttling at low loads, while eliminating throttling losses entirely at more moderate loads.

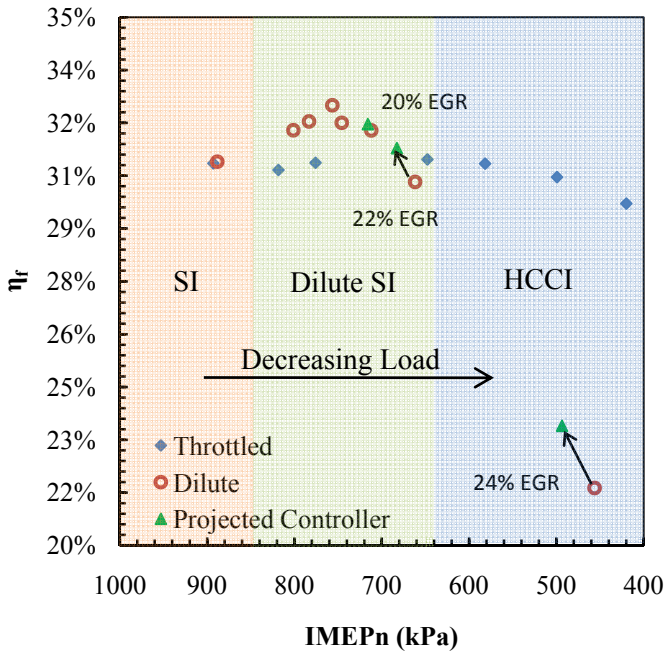

Figure 3.14. Potential multi-mode engine operation

3.3.4. Effect of Spark Timing on Controller Effectiveness. Spark timing is another factor to consider when determining operating parameters for such a controller. The effect of spark timing on the dynamics of the cycle-to-cycle variations can be substantial, as detailed by Wagner (75). For example, consider the following two examples of lean engine operation at the same equivalence ratio. The difference in engine behavior is illustrated with the use of return map plots. In Figure 3.15, a return map is shown for an equivalence ratio of 0.725, with spark timing set at MBT. There are some outliers, primarily misfires, but most of the data are clustered at a steady fixed point. In Figure 3.16, a return map shows data for the same equivalence ratio, but a more retarded spark timing, equal to the MBT timing for stoichiometric operation. In that case, the dynamics are more complex, with values other than misfires and those at the desired set point, and the COV is much higher.

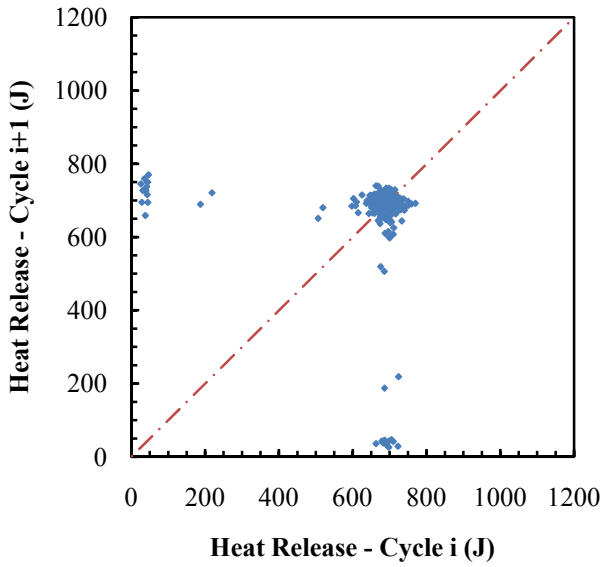

Figure 3.15. Return map for φ = 0.725, MBT spark timing

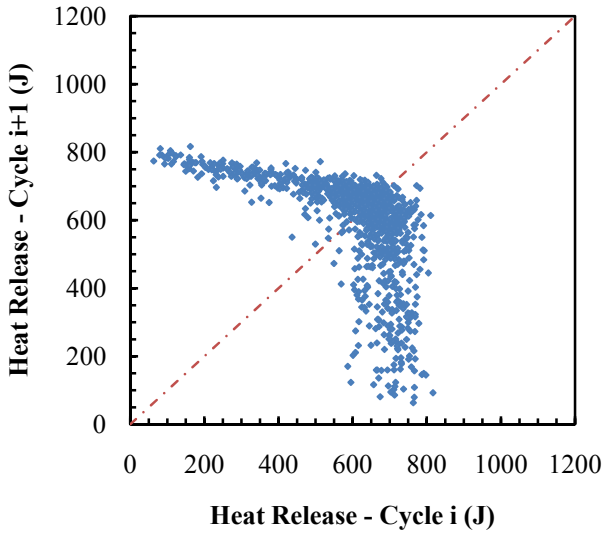

Figure 3.16. Return map for φ = 0.725, base spark timing

This effect is due to the misfire and partial-burn dilution limits described by Quader (11), where with the more retarded spark timing (nearer TDC), the partial-burn limit is being encountered, while with the more advanced spark timing, the misfire limit is dominant. This idea is illustrated in Figure 3.17.

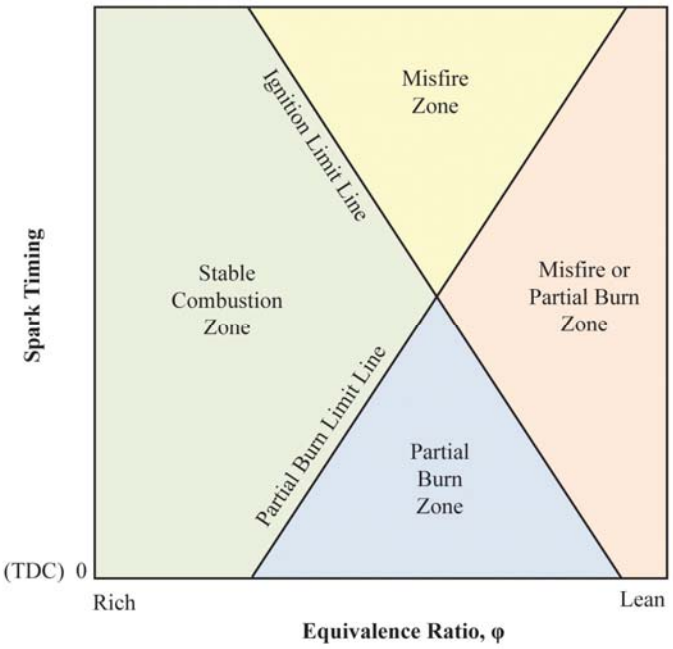

Figure 3.17. Zones of stable and unstable operation in lean mixtures

As shown in Table 3.4 for an equivalence ratio of 0.7, at MBT timing (60° BTDC) , where most of the variations are individual misfires, and low-high/high-low patterns of two cycles will therefore be common, the 8-2 analysis actually improves the fuel conversion efficiency more than the 2-6 analysis. For this advanced spark timing, the ignition limit shown in Figure 3.17 is dominant. However, for the more retarded base spark timing (15° BTDC), a controller considering only 2 cycles would not be able to effect noticeable change, while one considering longer sequences could potentially yield

nearly a 20% efficiency improvement over the uncontrolled case. For this retarded spark timing, the partial burn limit is encountered.

Table 3.4. Projected improvement with control at different spark timings

	Actual η_f	Cleaned Data 2-6		Cleaned Data 8-2	
		η_f	% Change	η_f	% Change
MBT	28.61%	29.51%	3.04%	30.86%	7.29%
Base Timing	14.30%	17.76%	19.48%	14.30%	0.00%

Thus, it is important to choose the spark timing appropriately in order to make best use of a prospective controller. With a controller design that considers a sufficient number of prior cycles, it can be desirable to deviate from MBT timing in order to move into a mode where the dynamics of the cyclic variability are more suitable for control.

Note also in Table 3.4 that the fuel conversion efficiency for the more advanced MBT timing is higher than for the more retarded base timing. When operating in the partial burn zone, the reduced burn rate causes a drop in efficiency. Therefore, the improved performance of the controller in these conditions must be balanced with the inherently lower efficiency of combustion to find the optimal spark timing for a given dilution level.

3.4. CONCLUSIONS

An estimation of the potential gains in efficiency that could be seen with an effective controller due to the reduction in cyclic dispersion is accomplished by filtering out cycles that contribute to repeating deterministic patterns in the sequence of symbolically categorized data. This projection matches well with the actual behavior of a controller at moderate dilution levels, and indicates that with improved control, the dilution limit can be extended, and substantial gains in efficiency can be obtained for dilute SI engine operation.

The analysis also indicates that spark timing should be considered when developing such a controller or planning multi-mode engine operation schemes, since it can have a dramatic affect on both the dynamics of the cyclic variations and the potential effectiveness of the controller, and a further effect on the combustion efficiency at high dilution levels.

4. SENSITIVITY OF ENGINE TO CONTROL INPUTS

4.1. FUEL MASS

While AI-based controllers such as the previously described NN controllers under development at Missouri S&T do not necessarily require explicitly defined relationships between perturbations to control inputs and the engine output, such information is still useful when evaluating and developing prospective controllers. It is difficult to determine whether a controller is giving reasonable inputs to the engine if the magnitude of a perturbation required to effect change in engine output is unknown. Accordingly, experiments were performed to determine the sensitivity of the engine to perturbations in fuel mass (the control input used by these controllers) at the dilute operating conditions of interest, where nonlinear response is expected.

4.1.1. Procedure. Fuel control sensitivity was tested for several dilution levels. For every case, the fuel was adjusted to maintain a stoichiometric equivalence ratio, and the spark timing was kept constant at the minimum spark advance for maximum brake torque (MBT) for no diluent, as determined by the peak in-cylinder pressure occurring at 16° after TDC. A sinusoidal variation was introduced onto the fuel injector pulse width, with a fixed period of 50 cycles. For each operating condition, the engine was operated for several hundred cycles to achieve steady state operation (verified by observing stabilization of exhaust temperatures), then 1000 consecutive cycles of in-cylinder pressure data were acquired.

These pressure data were then integrated from spark to EVO in a simple first-law analysis, neglecting heat transfer to the cylinder walls, as described previously, to determine the indicated heat release from combustion for each cycle. A fast Fourier transform (FFT) was performed on the heat release data. The power content at each frequency was calculated by multiplying the FFT of heat release with its conjugate. These power values were then examined for a dominant peak at the frequency corresponding to a period of 50 cycles and used to determine what percentage variation in fuel is necessary to cause a detectable variation in engine output at each dilution level.

4.1.2. Results. For each dilution level, data were collected for a constant fuel injector pulse width and also with sinusoidal variations having amplitudes of 1%, 2%, 3%, 4%, and 5% of the mean pulse width imposed on the fuel injector control signal. The engine was operated with no dilution and also with 13%, 16%, 18%, and 20% N_2 diluent concentration by mass.

For the non-dilute case, FFT power values are shown in Figure 4.1. For ease of reference, the horizontal scale has been converted from frequency to period, so a peak at 50 will be observable when the magnitude of the fuel variations is high enough. There is clearly no peak at 50 for the case with no fuel variations or for the 1% variation case. While there is a visible peak at this position for 2% and 3% variations in fuel, the peak is no higher than the background noise level of peaks occurring at many other frequencies. For 4% and 5% variations in fuel, the peak at a period of 50 is clearly visible and higher than at other frequencies.

At 13% diluent, the peak in the FFT power data at a period of 50 becomes apparent with a 2% variation in fuel, as shown in Figure 4.2. A harmonic at a period of 25 is also present at the higher amplitude variations. For 16% diluent, a fuel variation of 2% yields a peak in the FFT plot that is an order of magnitude higher than the background noise, as illustrated in Figure 4.3. For 1% fuel variation, a peak that is just slightly higher than other surrounding peaks can be observed, as can be seen in the enlarged view of Figure 4.6.

For 18% and 20% diluent concentrations, a variation of only 1% in fuel is sufficient to produce significant visible peaks, as shown in Figure 4.4 and Figure 4.5, or more clearly in Figure 4.6. In the 20% diluent case, some lower period (higher frequency) dynamics in the heat release data are also significant, even apart from the imposed variation in fuel. The coefficient of variation (COV) in heat release for the zero-fuel-variation data at this level of dilution is 18.5%, as the dilution level has risen to the point where cyclic variability is becoming severe. Some of the more complex dynamics observed by Sutton and Drallmeier (76) are also observed in this case.

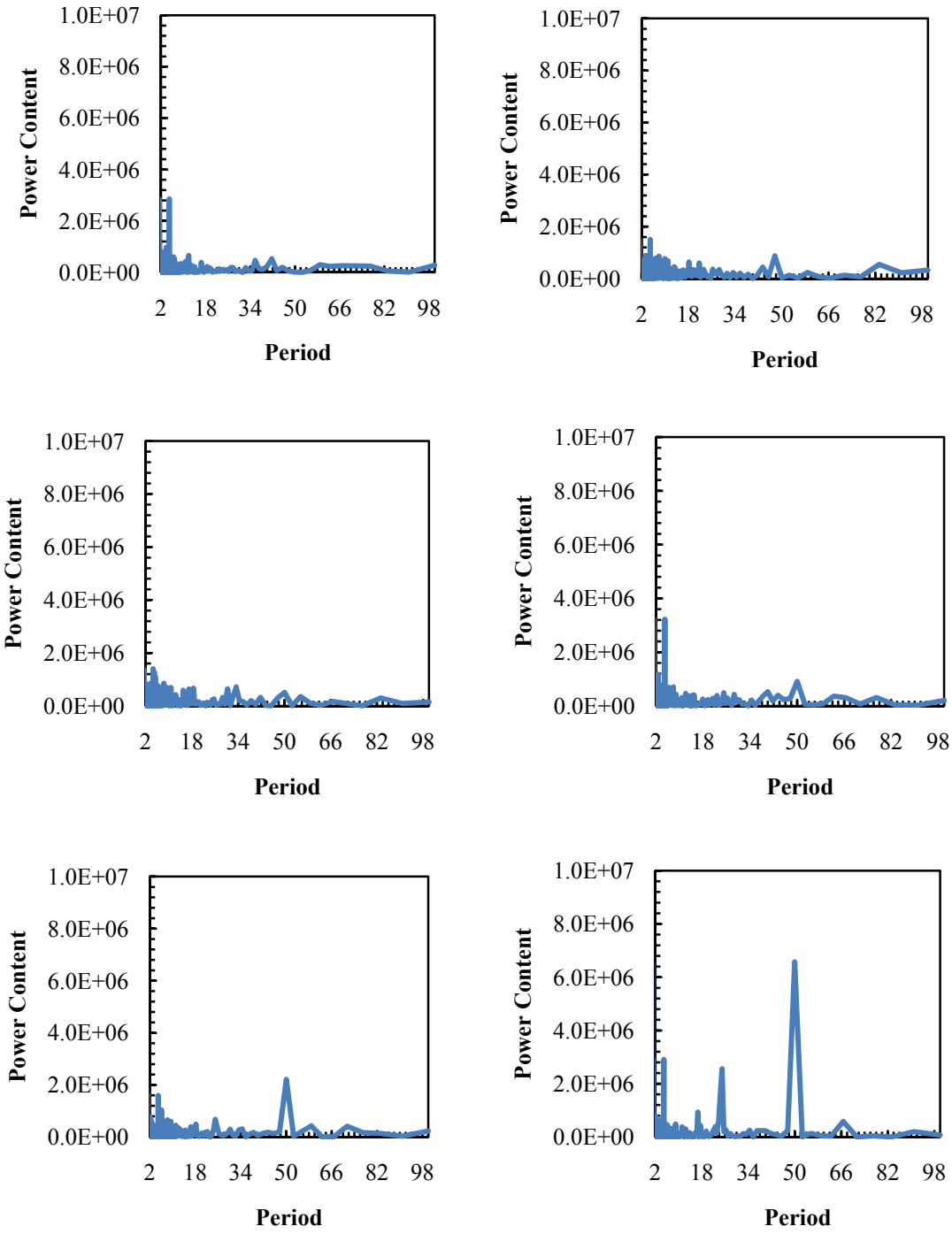

Figure 4.1. FFT power content for 0% diluent; fuel variation = 0%, 1% / 2%, 3% / 4%, 5%

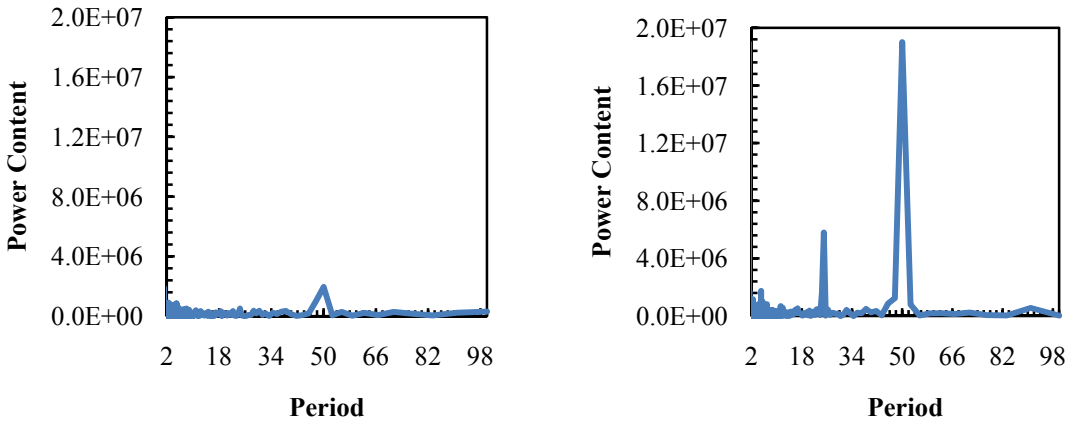

Figure 4.2. FFT power content for 13% diluent; fuel variation = 2%, 5%

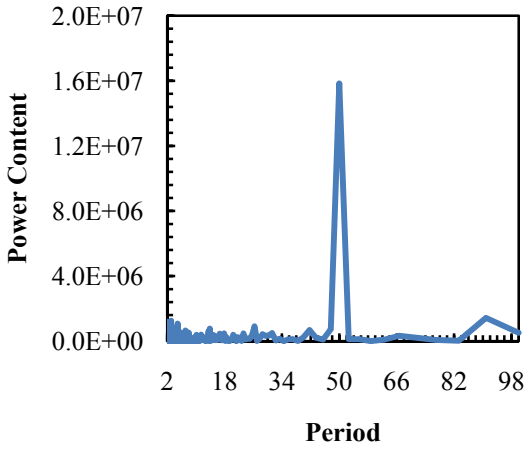

Figure 4.3. FFT power content for 16% diluent; fuel variation = 2%

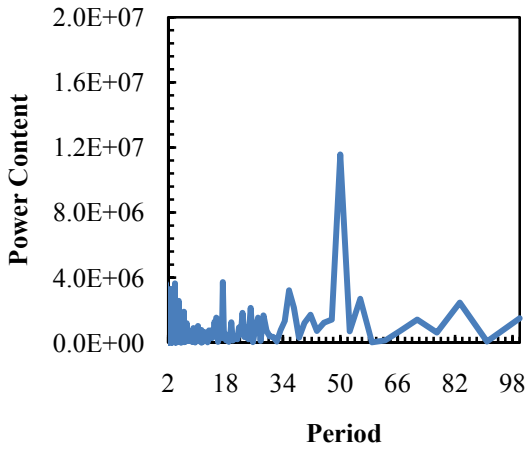

Figure 4.4. FFT power content for 18% diluent; fuel variation = 1%

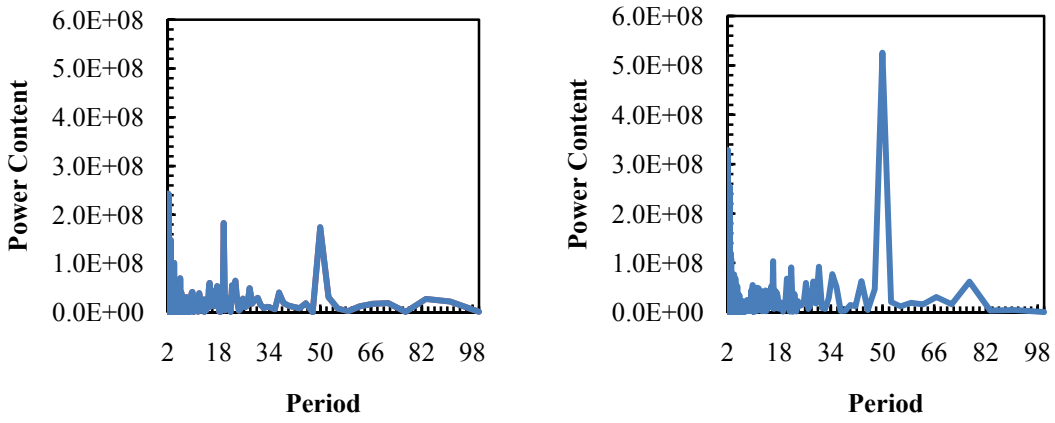

Figure 4.5. FFT power content for 20% diluent; fuel variation = 1%, 2%

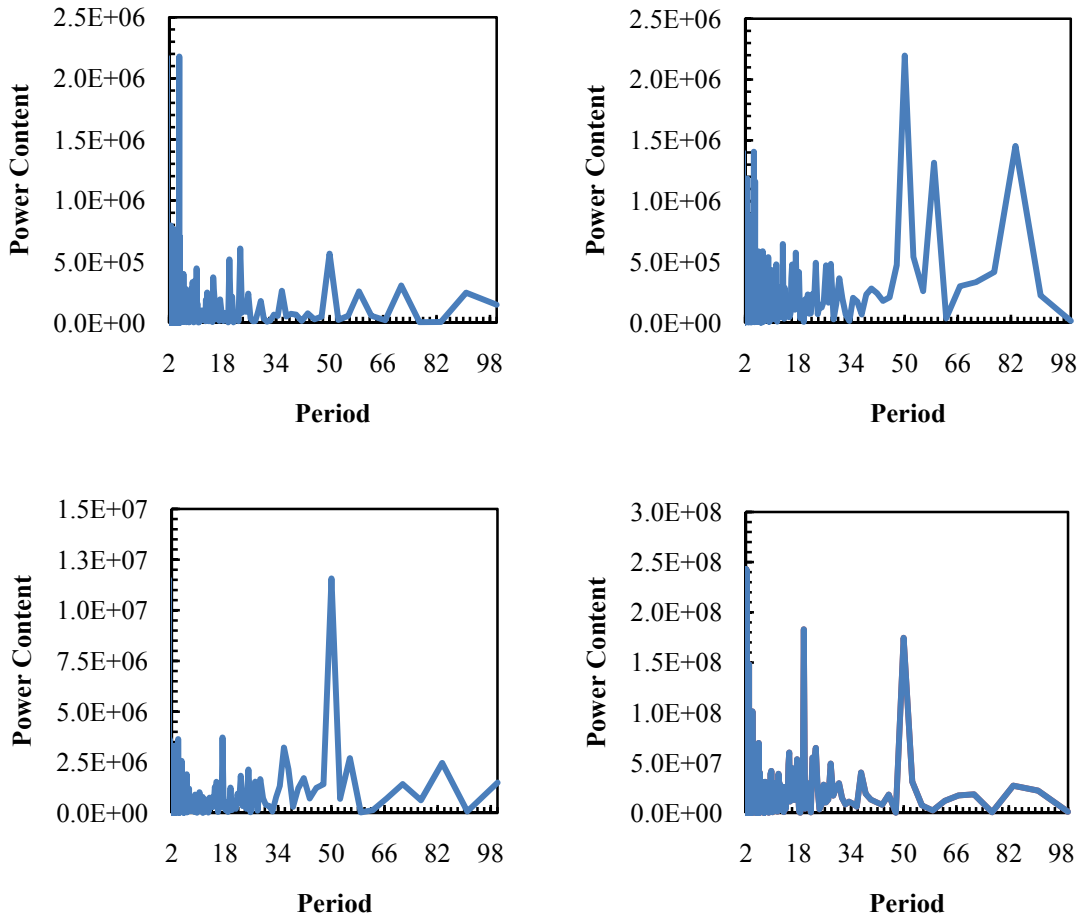

Figure 4.6. FFT power content with 1% fuel variation for 13%, 16% / 18%, 20% diluent. Note the differing vertical scale for each plot.

These increasingly nonlinear dynamics and increasingly deterministic variations as the level of dilution is increased cause combustion to be more sensitive to changes in equivalence ratio under these "strained" combustion conditions. This is observed in both the smaller fuel variation necessary to effect change at high dilution levels and the greatly increased magnitude of the changes observed. Note, for example, that in Figure 4.6, the scale for the 20% diluent plot is two orders of magnitude higher than that of the 13% and 16% diluent plots. This indicates that for higher dilution levels, the effect of perturbations to the fuel mass is much greater than at lower dilution levels, where the behavior is less nonlinear.

Changes in fuel injected under controlled operation with existing controllers under development (43; 44; 45; 46) are on the order of 1-3%. These results indicate that such a perturbation is be sufficient to effect a measurable change on the engine output at even moderate levels of dilution, and more so at higher levels of dilution.

4.1.3. Conclusions. Fuel pulse width variations were introduced at several dilution levels to determine the sensitivity of the engine to fuel perturbations as a potential cycle-to-cycle control input under dilute operating conditions. The magnitude of perturbation necessary to effect measurable change in engine output dropped from 4% for no dilution to less than 1% for high levels of dilution. This indicates that proposed controllers, which use perturbations on the order of 1%, should be able to effect change on the engine output with the observed level of changes to the control input.

4.2. OTHER CONTROL PARAMETERS

4.2.1. Spark Timing. An experiment was planned to perturb the spark timing control signal in the same periodic manner as the fuel pulse width signal. However, problems were encountered in implementing the necessary control hardware on the engine setup. The control signal for fuel pulse width has a negative polarity, such that it is normally high (5V), then drops low (0V) for the time that the fuel injector is supposed to open. The control signal required for the spark timing is a positive polarity, remaining low except for the time when the coil is charging and discharging. It is possible in firmware to configure the existing control hardware to output a positive polarity signal, but the hardware default is for negative polarity, and if the system crashed, the output of a sustained 5V signal would burn up transistor in the coil driver circuit.

Another BNC 500 delay generator box was thought to be a solution, since it is capable of inverting a negative polarity signal output by the control hardware. This would provide sufficient protection to the coil driver circuit and allow use of the existing control hardware to perform the spark timing sensitivity experiment. Unfortunately, it was discovered upon attempting this approach that the input impedance of the BNC 500 is only 50Ω, and this pulled the voltage of the control signal down below 2V, so that it was not sufficient to trigger the delay generator. An amplifier circuit will have to be added to the control hardware output to allow it to trigger the BNC 500 for a spark timing signal.

4.2.2. Dilution Level. Another control input that could be used is the EGR level, which could be controlled either through a VVT system for internal EGR (residual gas) control, or through changes to externally injected inert diluent. There is not, at present, hardware in place to support such a system. The second approach, however, could be implemented at reasonable cost. The same method of controlling bulk diluent would remain in place, with bottled gas being injected upstream in the surge tank. Another injection point for diluent would be added near the intake port, and a small perturbation on the diluent mass could be controlled by a fast-acting solenoid valve such as a fuel injector. Some amount of calibration would be required, but this would allow for fast, precise control of diluent concentration. With such as system in place, it would be possible to determine the sensitivity of the cycle-to-cycle dynamics to small changes in diluent concentration as a control input.

5. MULTI-CYCLE ENGINE MODEL

5.1. MODEL STRUCTURE

A multi-cycle engine model capable of predicting the nonlinear dynamical behavior of cyclic variability in dilute engine operation is desirable both for the direct physical insight that can be gained and for its utility as a tool for simulating and developing control strategies. This chapter describes such a model: in this case, the integration of a thermodynamic model and a turbulent burning rate model, with residual gas composition and temperature carried over from one cycle to the next, results in a combined model that shows similar cycle-to-cycle dynamics as are observed in actual engine operation.

5.1.1. Thermodynamic Model. A zero-dimensional, two-zone thermodynamic engine cycle model is used to simulate the thermodynamics. This model is based on the previous work of Caton (58; 59), and was completed in conjunction with Chakravarty, et al. (60) at Oak Ridge National Laboratory.

Model details are also given in reference (60). This model starts with a first law of thermodynamics basis, assuming the cylinder contents to be a closed system, and considering only the compression and power strokes. The pressure, temperature, and composition at the beginning of the compression stroke are given as an input to the thermodynamic model. Two zones are considered: burned and unburned gases. The pressure is homogeneous throughout the cylinder, and the composition and temperature are homogeneous within each zone. The unburned zone composition is frozen; the burned zone composition is assumed to be at equilibrium above 1600K and frozen once temperatures drop below 1600K during expansion after combustion has completed.

The molar fuel/oxygen ratio is used to define the composition of each zone. Rather than performing complex chemical equilibrium calculations and determining mixture properties online, lookup tables that were generated offline are used by the model to relate internal energy, temperature, pressure, and composition. Energy or temperature can be obtained by interpolating between values on the table given the other three properties.

The (nonlinear) conservation equations for mass and energy are integrated with respect to crank angle using an implicit second order scheme. The iterations to converge the implicit scheme are started by guessing the values of dependent variables at the new crank angle. Wall heat transfer is calculated from the (specified) wall temperature and a mass-averaged temperature of the contents of the cylinder using Equations 5.1 and 5.2:

$$\dot{Q} = A_h h\left(T_w - T_{avg}\right) \tag{5.1}$$

$$h = 0.02466\sqrt{\frac{pT_w}{1000}}\left(\frac{LN}{30}\right)^{0.33} \tag{5.2}$$

The heat transfer is apportioned between the two zones, and then the burning rate is estimated using the flame speed model. Given this burning rate, the appropriate mass is transferred from the unburned zone to the burned zone, and the associated energy transfer is estimated. The energy equation is used for each zone to calculate the new internal energy of each zone given the wall heat transfer rate and the mass/energy transfer rate between the zones. The lookup tables are then used to determine the new temperature for the unburned zone, and the cylinder pressure is calculated using Equation 5.3:

$$p = \frac{\lambda(m_u R_u T_u + m_b R_b T_b)}{V} + (1-\lambda)p \tag{5.3}$$

The burned zone temperature is then interpolated from the lookup tables using the new pressure and internal energy of the burned zone. The ideal gas law is used to calculate densities for each zone, and the volumes are determined from density and pressure. The iterations are continued until changes are sufficiently small (on the order of 1.0 x 10^{-6} for the results presented here).

5.1.2. Turbulent Entrainment Flame Speed Combustion Model. In order to determine the mass fraction burned at each step in the cycle, a combustion model is necessary. In Caton's implementation of a two-zone thermodynamic model (58; 59), a Wiebe function is used to directly relate crank angle to the burned mass fraction. In the

present model, a turbulent mass-entrainment flame speed model based on that of Tabaczynski et al. (63) is integrated to add the desired combustion physics. In the implementation reported in Chakravarty, et al. (60), the turbulent flame speed model was not fully implemented, but instead a wrinkled laminar flame approach was used, with the assumption that the turbulent flame area is seven times the laminar flame area. Here, the turbulent entrainment model is used.

To account for ignition, a small flame kernel of specified volume and temperature is instantly created at the crank angle specified for the spark timing. No ignition model is included; ignition is simply assumed to occur. Thus, complete misfires cannot yet be predicted, while partial burns can.

As the flame front propagates, new unburned mass entrained into the flame region is determined by Equation 5.4:

$$\frac{dm_e}{dt} = \rho_u A_f (u' + S_L) \tag{5.4}$$

In order to improve early problems with underprediction of the ignition delay, the u' term is only included once a specified burned mass fraction has been reached. The default value for this is 0.05. It is known that the initial flame development stage of combustion is characterized by laminar burning (9), so this approach does not lack a physical basis.

The flame area, A_f, must be determined from the percentage of the charge that has been burned. The burned volume is calculated from the burned mass using the ideal gas law with the current iteration's guesses for pressure and burned gas temperature. It is assumed that the combustion chamber is a right cylinder, neglecting any complications of cylinder head geometry. There are two engine geometries used: that of the CFR, which has a side-mounted spark plug, and that of the Ricardo engine, which has a centrally mounted spark plug in the cylinder head.

For the Ricardo engine geometry, it is assumed that the flame front remains hemispherical, and propagates from the center of the top face of the cylinder. An effective spherical radius is determined from the burned volume, and used to calculate the

surface area of the flame front. Until the flame has impinged on either the piston top or the cylinder walls, this calculation is trivial, and is given by Equations 5.5 and 5.6:

$$\rho = \left(\frac{3}{2\pi} V_b\right)^{\frac{1}{3}} \tag{5.5}$$

$$A_f = 2\pi\rho^2 \tag{5.6}$$

Once the flame front has impinged on any of the walls of the combustion chamber, the geometry becomes somewhat more complicated. The shape of the burned volume will be a sphere, less the area "cut off" by the cylinder wall or piston top. The surface area that must be calculated is that of the curved surface between the burned and unburned zones, but not including the surfaces where the burned zone contacts a wall of the combustion chamber. Three possible geometries exist. The flame radius could be larger than the cylinder height, thus impinging on the piston top, yet still smaller than the cylinder radius. The flame radius could be greater than the cylinder radius, but smaller than the cylinder height, and thus impinge on only the cylinder walls. Or, the flame radius could be greater than both the cylinder height and radius, impinging on both surfaces.

For the case where the flame radius is greater than the cylinder radius, the flame radius can still be analytically determined from the burned gas volume, by Equation 5.7, and the flame area calculated from Equation 5.8:

$$\rho = \sqrt{\frac{1}{\pi R}\left(V_b + \frac{\pi}{3} R^3\right)} \tag{5.7}$$

$$A_f = (2\pi\rho^2) - \left(2\pi\rho(\rho - R)\right) \tag{5.8}$$

If the flame radius is greater than the cylinder height, but less than the cylinder radius, then a closed form solution for the flame radius given the burned volume cannot be found. The equation for the volume of the burned zone in this case is Equation 5.9:

$$V_b = \pi\sqrt{\frac{4}{9}\rho^6 - \frac{1}{3}\rho^2 h^4 - \frac{1}{9}h^6} \tag{5.9}$$

Newton's method is used to solve for the flame radius that will give the burned volume, with the functions given in Equations 5.10 and 5.11:

$$f = \rho^6 - \frac{3}{4}\rho^2 h^4 - \frac{9V_b^2}{4\pi} - \frac{1}{4}h^6 = 0 \tag{5.10}$$

$$f' = 6\rho^5 - \frac{3}{2}\rho h^4 \tag{5.11}$$

The flame surface area is then given by Equation 5.12:

$$A_f = (2\pi\rho^2) - \left(2\pi\rho\left(\rho - \sqrt{\rho^2 - h^2}\right)\right) \tag{5.12}$$

For the final case, where the flame radius is greater than both the cylinder radius and the cylinder height, the solution for flame radius is again not attainable in a closed form. The burned volume is given by Equation 5.13:

$$V_b = \pi\left(\frac{2}{3}\rho^3 + \rho^2 R + \sqrt{\frac{4}{9}\rho^6 - \frac{1}{3}\rho^2 h^4 - \frac{1}{9}h^6} - \frac{1}{3}R^3\right) \tag{5.13}$$

The flame radius must again be solved for by Newton's method, with the functions given in Equations 5.14 and 5.15:

$$f = \frac{4}{5}R\rho^5 - R^2\rho^4 - \left(\frac{4V_b^2}{3\pi} - \frac{4}{9}R^3\right)\rho^3 + \left(\frac{2V_b}{\pi}R + \frac{2}{3}R^4 - \frac{1}{3}h^4\right)\rho^2 + \left(\frac{V_b^2}{\pi^2} - \frac{1}{9}h^6 - \frac{1}{9}R^6\right) = 0 \tag{5.14}$$

$$f' = \frac{20}{3}R\rho^4 - 4R^2\rho^3 - 3\left(\frac{4V_b^2}{3\pi} - \frac{4}{9}R^3\right)\rho^2 + 2\left(\frac{2V_b}{\pi}R + \frac{2}{3}R^4 - \frac{1}{3}h^4\right)\rho \tag{5.15}$$

The flame surface area is then calculated from Equation 5.16:

$$A_f = (2\pi\rho^2) - \left(2\pi\rho\left(\rho - \sqrt{\rho^2 - h^2}\right)\right) - \left(2\pi\rho(\rho - R)\right) \tag{5.16}$$

Given the different relationships between the effective flame radius and the burned zone volume and flame surface area, and the lack of closed form solutions for some flame radius values, it is necessary to make an initial guess for the flame radius, and iterate until a solution is reached that matches the burned zone volume.

For the side-spark CFR geometry, the intersection of a spherical flame front centered on the wall of a cylinder is even more complex. Since the primary interest is in the more commonly used center-spark geometry, a simplified model was implemented here, with a cylindrical flame front propagating from the wall to fill the cylinder.

Once the flame radius and flame surface area have been solved for in this manner, the rate of mass entrainment given by Equation 5.4 can be calculated.

The entrained mass is then burned at a rate proportional to the amount of unburned mass present in the flame area, as given by Equation 5.17:

$$\frac{dm_b}{dt} = \frac{m_e - m_b}{\tau} \tag{5.17}$$

The characteristic burning time, τ, is given by Equation 5.18:

$$\tau = \frac{l_m}{S_L} \tag{5.18}$$

The turbulence intensity is assumed to be initially proportional to the piston speed at the start of combustion, after which it is governed by the conservation of momentum, along with the integral scale, according to Equations 5.19 and 5.20:

$$l_i = (l_i)_0 \left(\frac{\rho_0}{\rho_u}\right)^{\frac{1}{3}} \tag{5.19}$$

$$u' = u'_0 \left(\frac{\rho_u}{\rho_0}\right)^{\frac{1}{3}} \tag{5.20}$$

For isotropic turbulence, the Taylor microscale is given by Equation 5.21:

$$l_m = \sqrt{\frac{15\nu l_i}{u'}} \tag{5.21}$$

where ν, the dynamic viscosity, is given by the correlation shown in Equation 5.22:

$$\nu = \frac{\nu_S \rho_S}{\rho_u} \left(\frac{T_u}{T_S}\right)^{0.76} \tag{5.22}$$

The flame speed correlation developed by Metghalchi and Keck (69) accounts for effects of temperature, pressure, and diluent concentrations on the laminar flame speed, and is determined by Equation 5.23:

$$S_L = S_{L0} \left(\frac{T_u}{T_0}\right)^{\alpha} \left(\frac{p}{p_0}\right)^{\beta} (1 - 2.1 Y_{dil}) \tag{5.23}$$

where S_{L0}, α, and β are given by Equations 5.24, 5.25, and 5.26:

$$S_{L0} = B_M + B_2(\varphi - \varphi_M)^2 \tag{5.24}$$

$$\alpha = 2.18 - 0.8(\varphi - 1) \tag{5.25}$$

$$\beta = -0.16 + 0.22(\varphi - 1) \tag{5.26}$$

Values for B_M, B_2, and φ_M are fuel-dependent, and can be found in Turns (70) or Heywood (9) for a variety of fuels.

These equations collectively determine the change in mass entrained and burned mass at each crank angle step relative to the previous value. The new burned mass is passed back to the thermodynamic model, which calculates new temperatures, a new

pressure, and a new volume for each zone. The model iterates to convergence for each crank angle step, then moves on to the next crank angle until the cycle is complete.

Heat release can either be determined directly from the thermodynamic energy increase or calculated in the same manner as for experimental data, based on a first-law analysis of cylinder pressure. As with the Daw model (26), stochastic variations in the form of Gaussian noise can be introduced onto input parameters in order to capture the effects of random or higher-order dynamics.

5.1.3. Cycle-to-cycle communication. The above thermodynamic and turbulent burning rate models are used to simulate each individual engine cycle. In order to capture the cyclic variations, an approach inspired by the model of Daw, et al. (26) is used. The feed-forward communication mechanism in the engine is the residual gases that are not exhausted after each cycle. Normally distributed noise on input parameters accounts for stochastic variations caused by turbulence and other such factors.

The temperature of the residual gases is assumed to be the mass-averaged temperature of the burned and unburned gases at the end of the previous cycle. A residual fraction is specified as an input parameter, and mixed in proper proportion with the fresh intake charge (of specified composition). The temperature of the new mixture of fresh intake charge with residual gases is iteratively determined from the property tables given the pressure, composition, residual fraction, and initial guesses of temperature from the fresh charge temperature and residual gas temperature.

Gaussian noise can be imposed on input parameters to account for stochastic variations. Typically, a noise level of approximately 1% on the equivalence ratio gives return maps with a similar level of noise to those observed experimentally. Noise can also be imposed on a number of other input parameters, including the residual fraction and EGR level.

5.2. SINGLE CYCLE VALIDATION

In order to verify that the model produces accurate simulations of engine behavior, the modeled pressure trace for a single typical cycle at near-stoichiometric conditions was compared with experimental data. To provide a typical cycle pressure trace for comparison, 1000 cycles of experimental pressure data were averaged together.

Since the model does include feed-forward communication between consecutive cycles, the first few cycles simulated will be prone to error due to imperfect initial assumptions regarding the residual gas composition and temperature. Model heat release results for zero parametric noise approach a stable value within 5 cycles, as shown in Figure 5.1, and are indistinguishable within $1x10^{-6}$ after 10 cycles. In order to avoid any artifacts of model initialization, the twentieth cycle simulated was used for comparison with the averaged experimental pressure data. The first ten simulated cycles are not included in any results documented here.

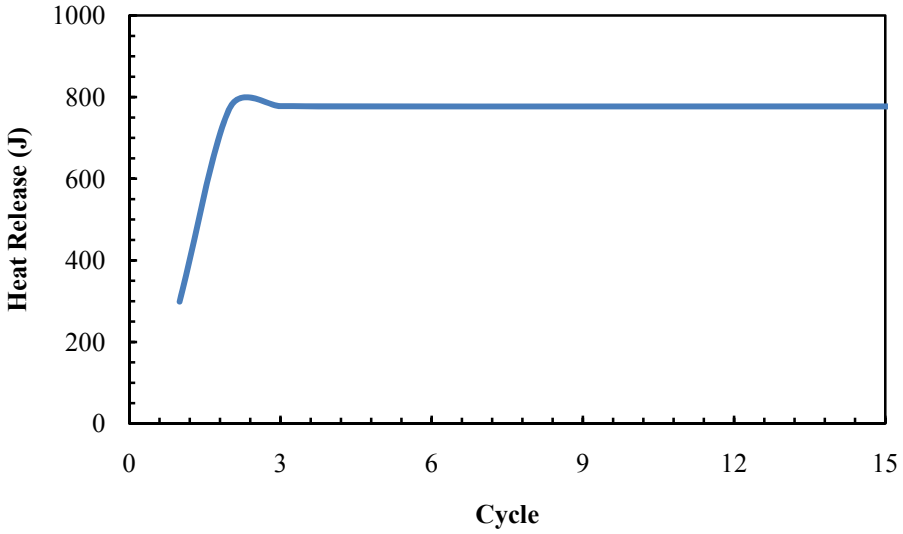

Figure 5.1. Stabilization of simulated heat release value

The thermodynamic portion of the model was initially validated by considering a motored cycle. The MAP pressure was input, along with an estimate of the cylinder wall temperature; other parameters apart from constants for the engine are not relevant if no combustion occurs. Figure 5.2 shows the model simulation and experimentally measured

cylinder pressure for part-throttle, motored operation. The prediction is very good, with almost no distinguishable difference between the measured data and the model results.

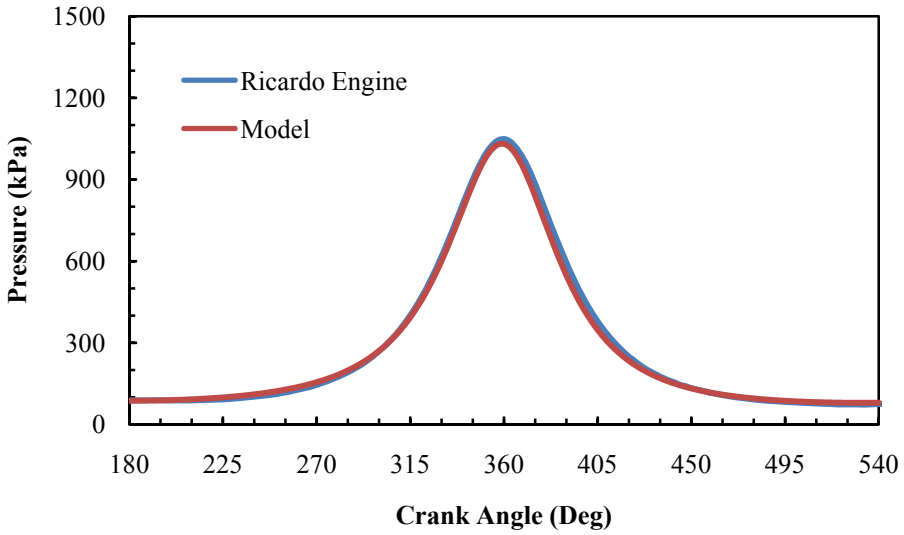

Figure 5.2. Motored pressure trace comparison

A near-stoichiometric equivalence ratio of 0.9 was chosen as an initial test case. Figure 5.3 shows both the model prediction and experimental data from the Ricardo engine at the same operating conditions. The initial pressure rise due to compression is well predicted, but the model underpredicts the ignition delay slightly. Peak pressure location matches well, though the magnitude is noticeably overpredicted. The drop-off in pressure seen after about 425° appears to be due to the simplified heat transfer correlations overpredicting the cooling during expansion.

Unfortunately, this drop in pressure towards the end of the cycle precludes the option of using the same P-V based heat release calculation as is used for experimental data analysis. The excessive pressure drop actually causes negative heat release values to be reported in many cases. Instead, heat release calculated from the integration of the

rate of change of internal energy is reported herein. These values will not numerically match the experimental data, but the trends and dynamics are the same.

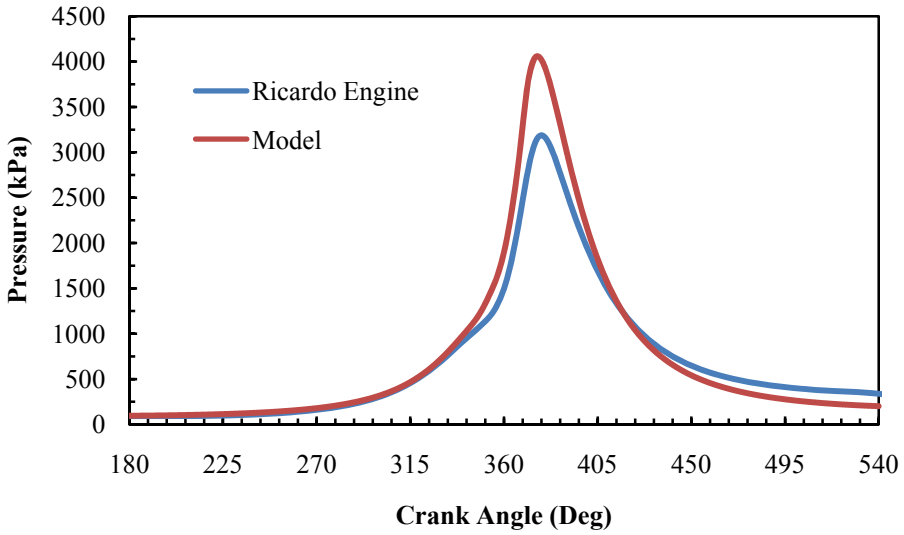

Figure 5.3. Comparison of model to experimental pressure trace for φ=0.9, MBT timing

Since one key advantage of this model over previous models that simulated cyclic variations is its ability to take timing effects into consideration, a reference case with the timing retarded from MBT for the actual equivalence ratio in use (19.2° BTDC) back to a base timing of MBT for the stoichiometric condition (15° BTDC) was also considered, and is shown in Figure 5.4. In this case, the ignition delay is again too small, and the overprediction of peak pressure is also reduced. The trend of decreased and delayed peak pressure compared to the MBT case is correct.

Leaner equivalence ratios were also evaluated on a single-cycle, averaged basis. As the equivalence ratio is reduced, decreased peak pressure and slower burn rates are observed. Figure 5.5 shows the averaged experimental pressure trace and the model-predicted pressure trace for an equivalence ratio of 0.85.

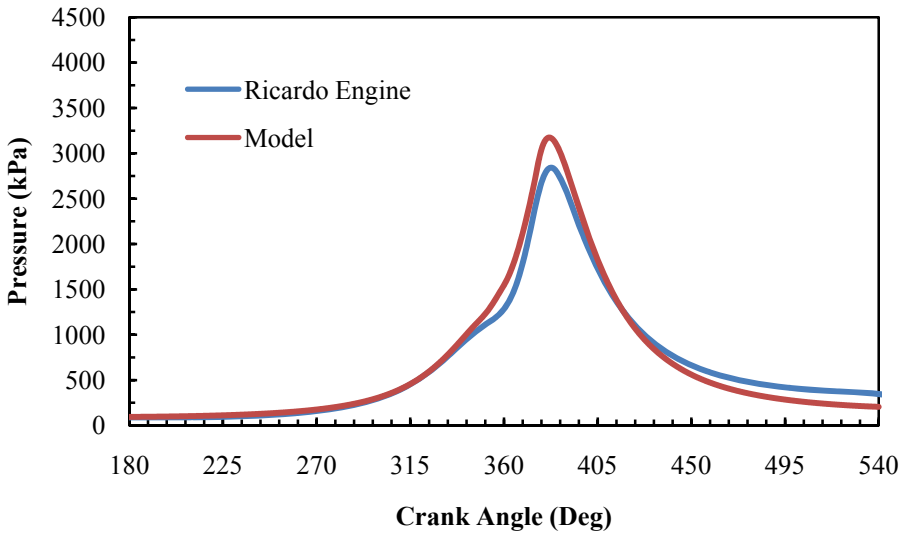

Figure 5.4. Comparison of model to experimental pressure trace for φ=0.9, base timing

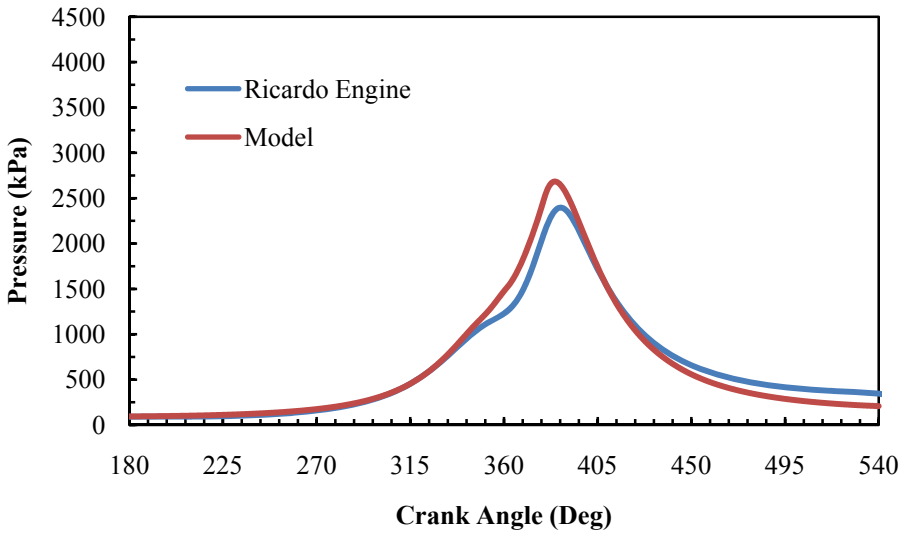

Figure 5.5. Comparison of model to experimental pressure trace for φ=0.85, base timing

As the equivalence ratio is further reduced, the trend of decreased and delayed peak pressure continues, as can be seen in Figure 5.6 and Figure 5.7 for equivalence ratios of 0.8 and 0.75, respectively.

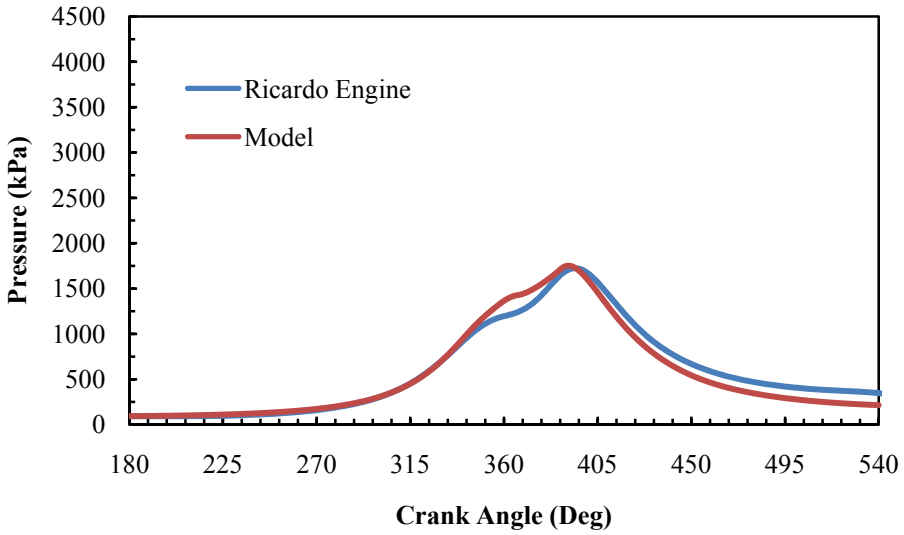

Figure 5.6. Comparison of model to experimental pressure trace for φ=0.80, base timing

The model consistently has a somewhat greater pressure rise early in combustion than is observed experimentally, and consistently predicts too great a pressure drop during expansion, but the trends with respect to spark timing and equivalence ratio are correct. For leaner equivalence ratios, the level of cyclic variability becomes significant, so single-cycle comparisons to experimental results cannot give an indication of model performance.

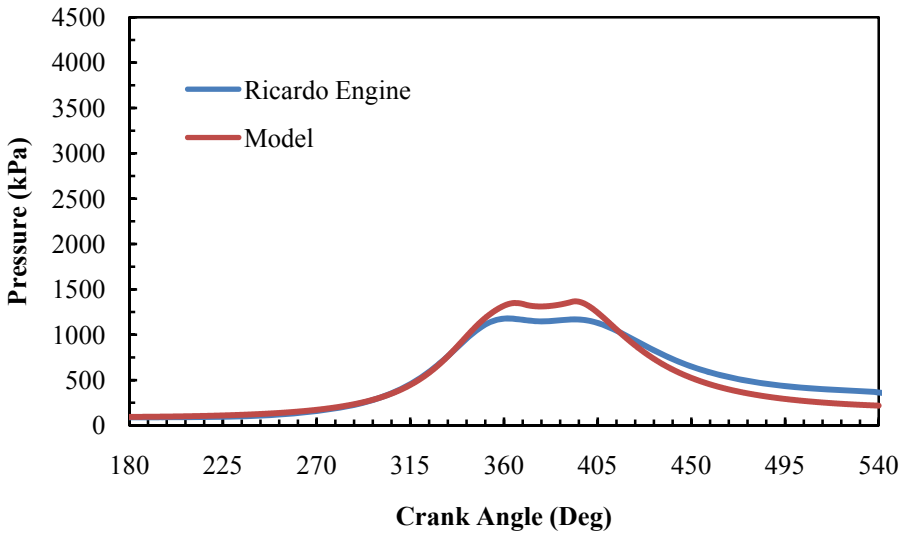

Figure 5.7. Comparison of model to experimental pressure trace for φ=0.75, base timing

While there are clearly imperfections in a model this simple, it successfully predicts the correct trends of changes in combustion duration and peak pressure as both composition and spark timing are varied. With the inclusion of cycle-to-cycle communication through residual gases, this should enable simulation of the dynamics of the cyclic variability encountered at higher dilution levels when the partial burn limit is relevant. Additionally, since the envisioned control strategies do not rely on a detailed online model, but this model is instead intended to predict the reaction of an engine to a controller that is itself capable of learning online, the correct trends shown with changes in both composition and timing will be useful for control simulations, so long as the dynamical response is accurate.

5.3. MULTI-CYCLE RESULTS

The primary motivation for the compilation of this model was to provide more physically detailed multi-cycle simulations than were previously possible. Previous single-cycle models could, when given sufficient time, provide accurate simulations of an average engine cycle, and previous multi-cycle models could simulate cycle-to-cycle dynamics based upon mathematical equations, but lacked the ability to predict behavior

based on changes to many physical parameters. Here, a simple physical model is used to simulate cycle-to-cycle dynamics based on thermodynamics and combustion physics.

5.3.1. Experimental bifurcation sequences. The qualitative change in the dynamics of cyclic variations as dilution is increased can be seen when observing a bifurcation diagram. Figure 5.8 shows such a bifurcation sequence for the Ricardo engine at part throttle conditions, from Sutton (77). For each equivalence ratio, data are collected for a number of consecutive cycles, and the heat release value for each of these cycles is then plotted with respect to that equivalence ratio, forming a vertical slice of the plot. Each point plotted is the heat release value for a single engine cycle. By repeating this for a large number of closely spaced equivalence ratios, the entire diagram can be constructed.

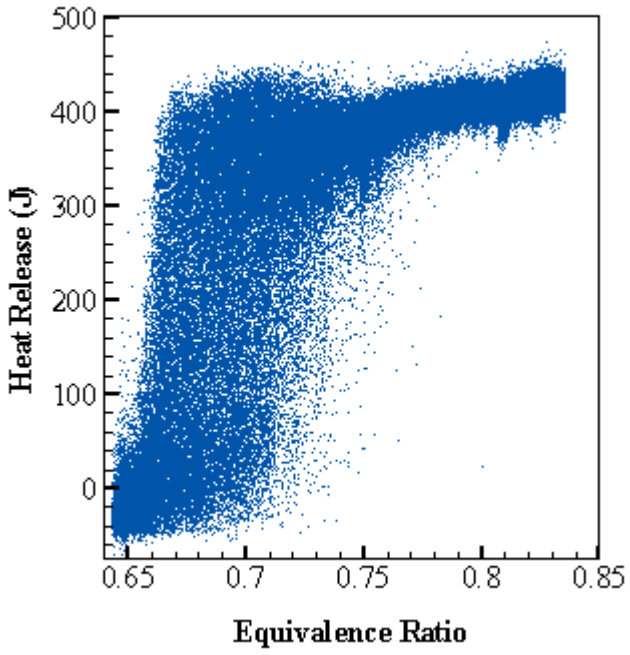

Figure 5.8. Bifurcation diagram for Ricardo engine

It is apparent that initially, as the equivalence ratio is decreased, a slight linear decrease in heat release occurs as well. However, once some threshold is reached, the dynamics qualitatively change, and the majority of the cycles are either very low-energy or very high-energy events, contributing to the dark bands at the top and bottom of the plot. This split is known as a period-doubling bifurcation. Wagner (75) and Sutton (77) characterized this behavior in detail for a variety of dilute operating conditions.

This behavior is independent of engine design, and common to SI engines operating with high levels of charge dilution. Figure 5.9, from Chakravarty, et al. (60), shows such a plot for experimental data collected by Wagner (75) on the CFR engine configured in such a way as to have a high residual fraction of 0.28. At this higher residual fraction, more complex dynamics are observed.

Wagner, et al. (38) later developed a method of filtering much of the stochastic noise from these plots by using nonlinear statistical regression to fit map functions to the return map for each operating condition. These map functions have mathematical fixed points, which can then be plotted, showing the underlying nonlinear dynamics. Figure 5.10, also from Chakravarty, et al. (60), illustrates the results of this method when applied to the data that was used to generate Figure 5.9. The initial period-doubling bifurcation is clearly seen at the right. At the left of the plot, for leaner equivalence ratios, the dynamics become higher order and possibly chaotic.

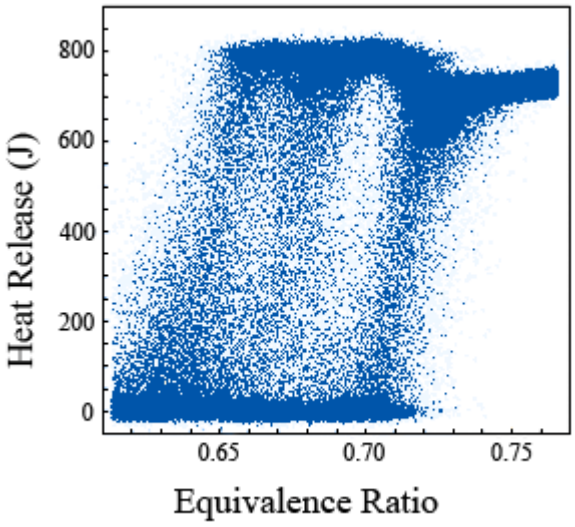

Figure 5.9. Bifurcation diagram for CFR engine with residual fraction of 0.28

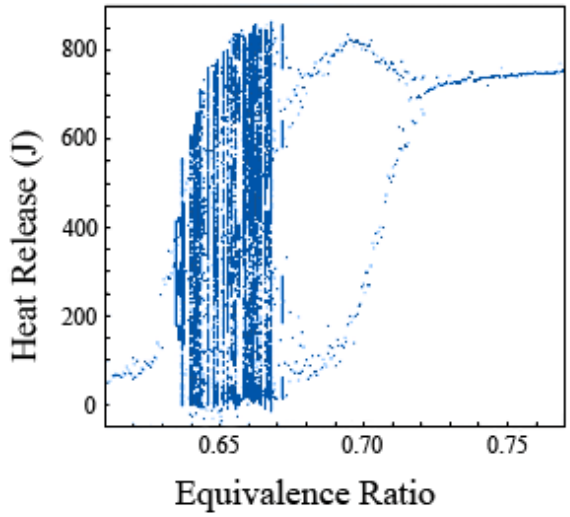

Figure 5.10. Deterministic component of bifurcation diagram shown in Figure 5.9

5.3.2. Model bifurcation sequences without turbulent entrainment. As described previously in Chakravarty, et al. (60), this model was first implemented with the effects of turbulence accounted for by a multiplicative factor of seven on the laminar flame area. In this form, the model was able to reproduce the basic features of this bifurcation sequence, with the initial period doubling, followed by a transition to higher order dynamics, as shown in Figure 5.11. A slightly higher residual fraction of 0.34 was required to achieve this result, compared to the experimental operating conditions, and the match is imperfect with regards to the exact location of the change in dynamics, but the qualitative similarity is clear.

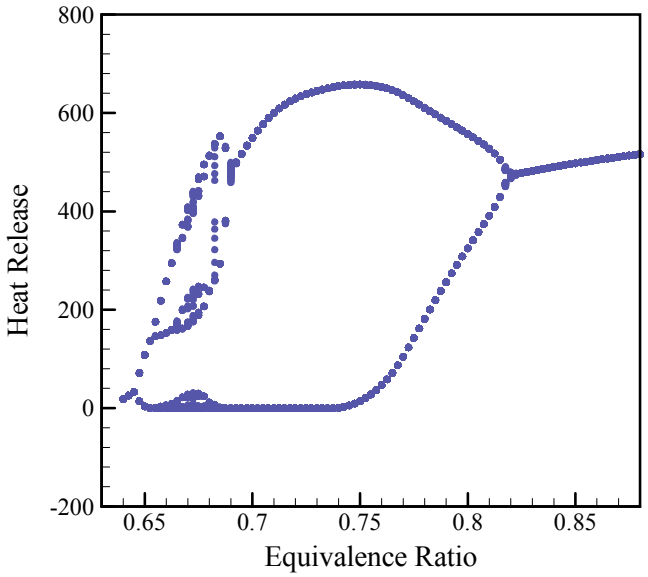

Figure 5.11. Bifurcation diagram for model at residual fraction of 0.34 with no noise

At lower residual fractions, the high order dynamics that were shown above for very low equivalence ratios do not occur. Figure 5.12 shows the deterministic component of the bifurcation sequence for experimental engine data. After the initial

period-doubling bifurcation, the two "branches" converge again as combustion ceases altogether. The negative values for heat release are due to the simple model used to calculate it from measured pressure data, which do not account for any losses such as those caused by heat transfer to the cylinder walls.

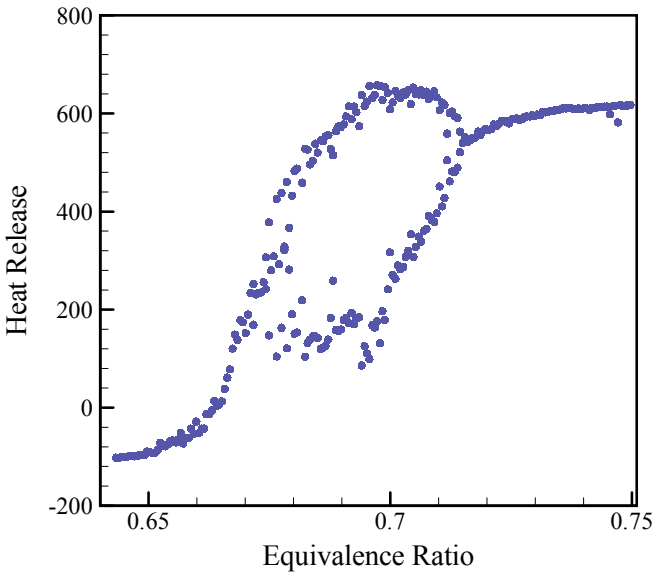

Figure 5.12. Deterministic component of bifurcation sequence for CFR engine with residual fraction of 0.14

The model correctly predicted this simpler dynamical behavior as well, for this residual fraction, though it is shifted somewhat with regards to the equivalence ratio at which the bifurcation will begin, predicting a leaner equivalence ratio at the onset of this transition in the dynamics. Figure 5.13 shows the bifurcation sequence generated by the model at these conditions.

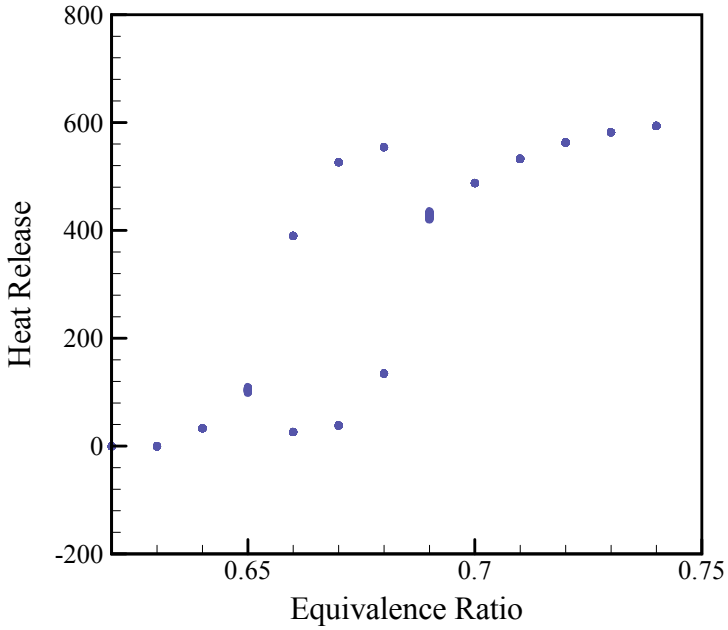

Figure 5.13. Bifurcation sequence for model at residual fraction of 0.14 with no noise

While this form of the model successfully generates bifurcation diagrams that are very similar to those obtained from experimental data, it is not able to generate return maps of the same form as those observed experimentally, as the simple model of Daw, et al. (26) was. The turbulent entrainment model was implemented to replace the assumed factor of seven effect of flame front wrinkling on the burn rate.

5.3.3. Model results with turbulent entrainment. When the turbulent entrainment submodel is included, more physical insight can be gained by examining the state variables of individual cycles that behave differently. For example, at an equivalence ratio where a bifurcated pattern of alternating high and low energy combustion events is occurring, the high and low energy cycles can be examined in detail. The following sequence of plots will compare the high and low energy cycles generated by the model at $\varphi = 0.68$ with no parametric noise.

To start, it is to be expected that for cycles with different heat release levels, the pressure trace would differ. For the two cycles shown in Figure 5.14, the higher energy cycle has a heat release value of 1061 J, while the lower energy cycle has a heat release value of 100 J.

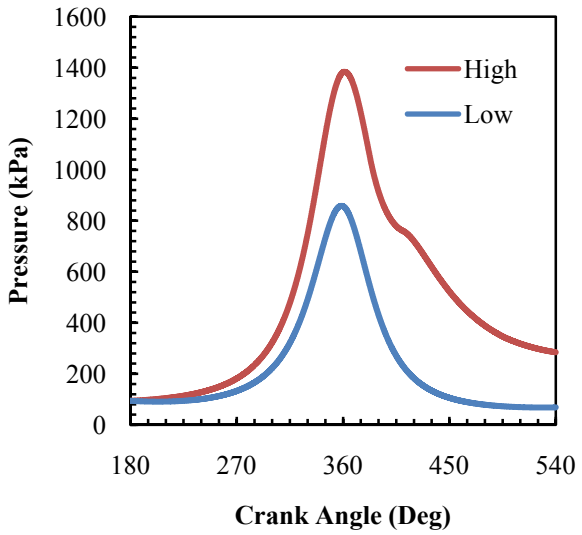

Figure 5.14. Model pressure trace comparison for high and low energy cycles at φ=0.68

Further variables other than pressure can also be examined, though. For instance, the temperature of the unburned gases varies significantly early in the cycle, due to the effect of the residual gases from the previous cycle, as seen in Figure 5.15. Data for a simulation at a near-stoichiometric condition of φ = 0.9 are also shown for reference. Note in Figure 5.15 that the low-energy cycle has a much higher initial temperature due to the carryover effect of the residual from the preceding high-energy cycle. The temperature of the burned gases, on the other hand, shows the opposite trend: the high-energy cycle in Figure 5.16 exhibits higher flame temperatures.

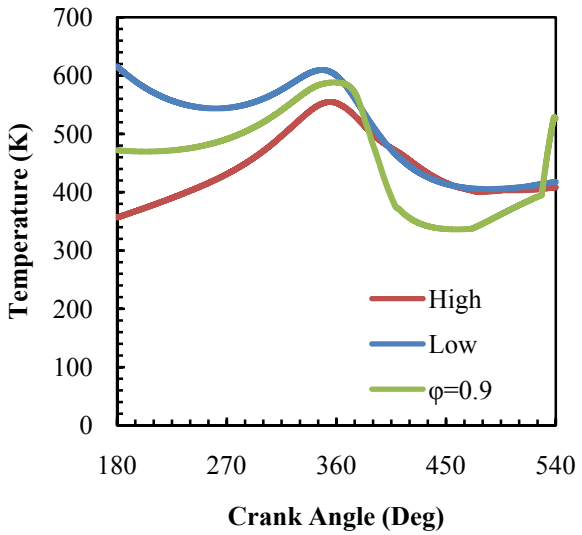

Figure 5.15. Model unburned gas temperatures for high and low energy cycles at φ=0.68

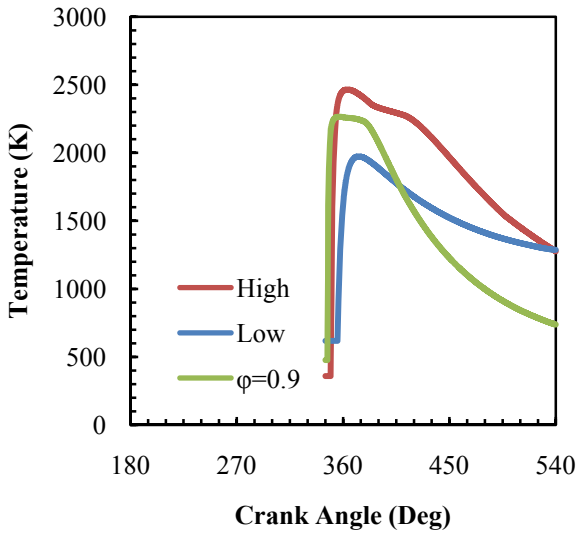

Figure 5.16. Model burned gas temperatures for high and low energy cycles at φ=0.68

If the mass-averaged temperature of the entire cylinder is considered, as in Figure 5.17, the difference is more apparent. The high-energy cycle comes much closer to completion of combustion, and has a large burned mass fraction towards the end, as shown in Figure 5.18, so the mass-averaged temperature is much higher, since the amount of burned mass is not negligible.

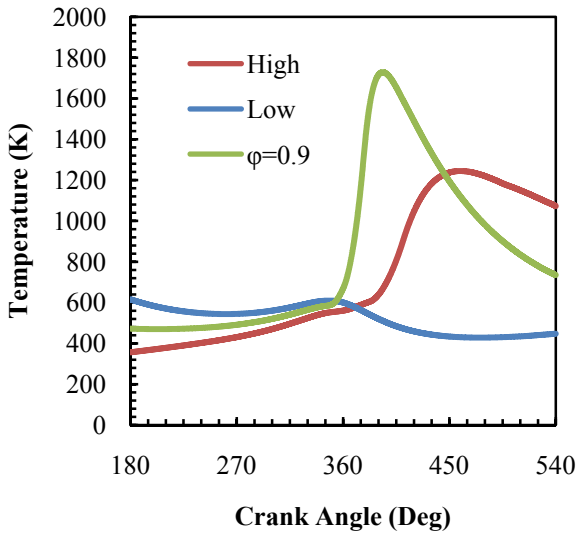

Figure 5.17. Model bulk gas temperatures for high and low energy cycles at φ=0.68

The higher temperatures and pressures present in the cylinder during the higher energy cycles serve to amplify the already greater initial flame speed that is caused by the higher internal equivalence ratio due to the unburned fuel present in the residual gases. This effect can be seen in Figure 5.19, where the laminar flame speed for the high-energy cycle is rougly twice that for the low-energy cycle.

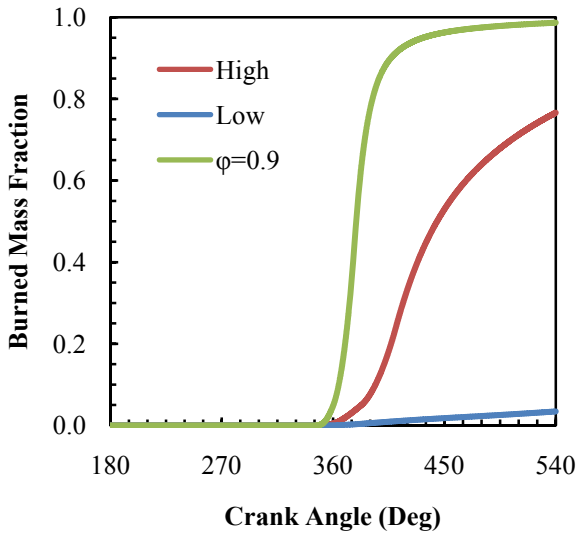

Figure 5.18. Model burned mass fractions for high and low energy cycles at φ=0.68

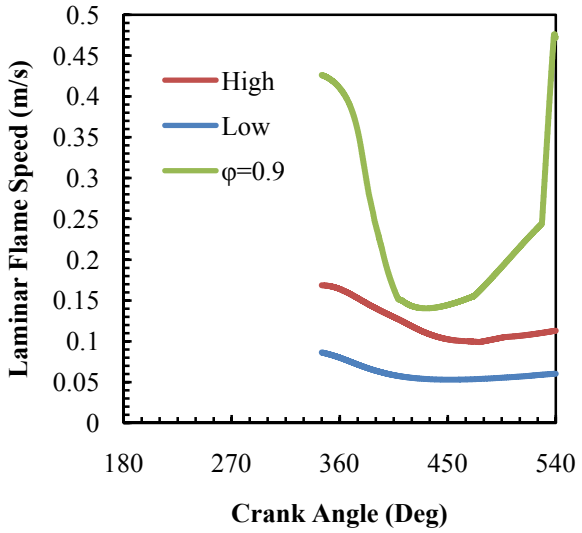

Figure 5.19. Model laminar flame speeds for high and low energy cycles at φ=0.68

The rate at which the charge is burned is based on the laminar flame speed, and additionally the turbulence intensity and flame front surface area. The overall mass burning rate, $\frac{dm_b}{dt}$, is shown in Figure 5.20. Here, the differences are even more amplified, showing that not only is the nonlinear dependence of the laminar flame speed on composition important, but also that the importance of the nonlinear dependence of the burning rate on flame speed and turbulence intensity makes the system even more sensitive to small initial changes.

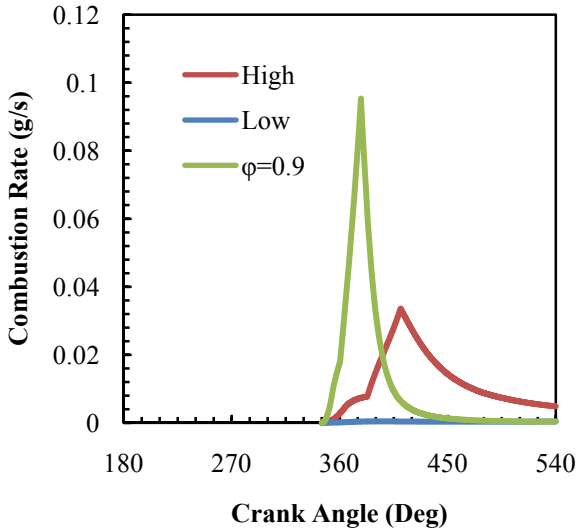

Figure 5.20. Model mass burning rate for high and low energy cycles at φ=0.68

These insights gained from examining internal system variables are not possible with simpler models such as that of Daw, et al. (26) or even the form of this thermodynamic model presented in Chakravarty, et al. (60) that lacked the turbulent entrainment equations.

5.3.4. Model bifurcation sequences with turbulent entrainment. This combined model, with the turbulent entrainment burning submodel included, is also capable of generating similar bifurcation sequences to those observed in experiments, though it is necessary to specify a much higher residual fraction. Figure 5.21 shows the bifurcation diagram for a residual fraction of 0.38 and base spark timing of 15° BTDC, with respect to equivalence ratio. While the residual fraction is higher, the dynamics are the same as observed engine behavior. Figure 5.22 shows a bifurcation plot for the same operating parameters, but with 10% random noise on the input equivalence ratio. The random noise imposed on input parameters is used to account for the stochastic component of cycle-to-cycle variations in engine output, just as it was in the model of Daw, et al. (26). Here, though, the deterministic variations are predicted by models of thermodynamics and combustion physics, rather than by an empirically derived mathematical equation.

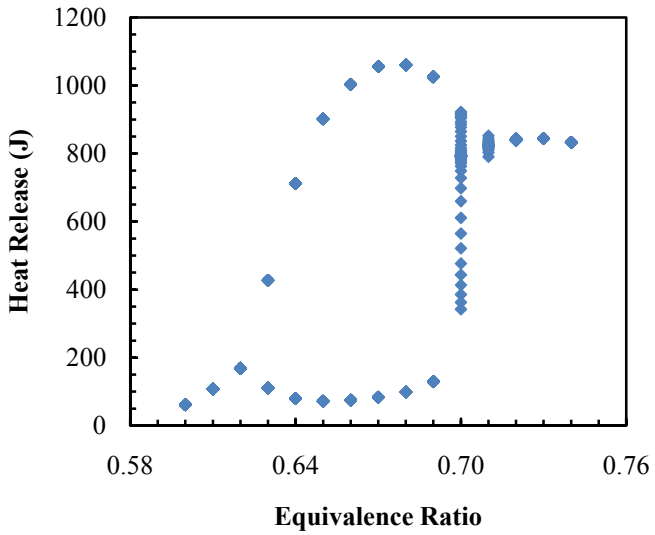

Figure 5.21. Bifurcation of modeled heat release with varying equivalence ratio for residual fraction of 0.38 with no parametric noise

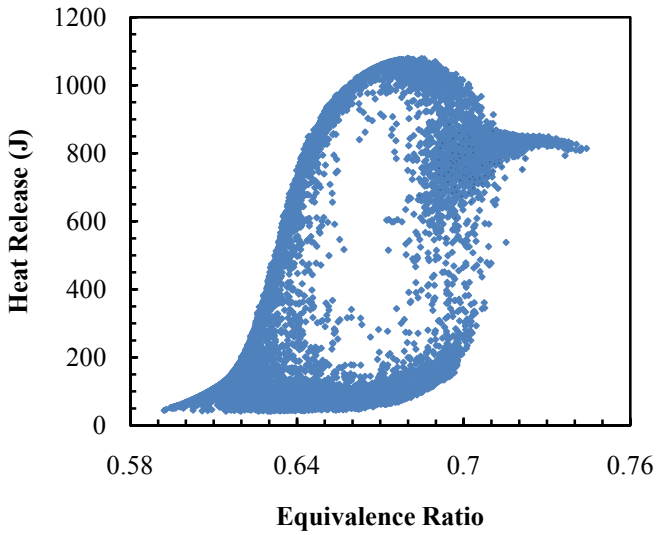

Figure 5.22. Bifurcation of modeled heat release with varying equivalence ratio for residual fraction of 0.38 with 10% parametric noise

Higher order dynamics at higher residual fractions are also observed, though this is again shifted to higher residual fractions than observed in the engine. The bifurcation sequence for a residual fraction of 0.60 is shown in Figure 5.23. In addition to the similar bifurcations when the input composition is modified at a constant residual fraction, it is also possible with the model to hold the input composition constant while varying the residual fraction. This result shows the period-doubling bifurcation as the residual fraction is increased, and the higher order dynamics at extremely high residual levels, as seen in Figure 5.24.

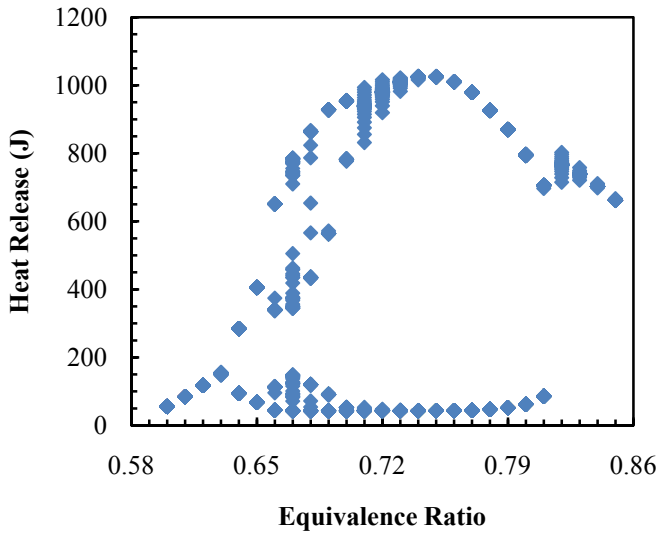

Figure 5.23. Bifurcation of modeled heat release with varying equivalence ratio for residual fraction of 0.60 with no parametric noise

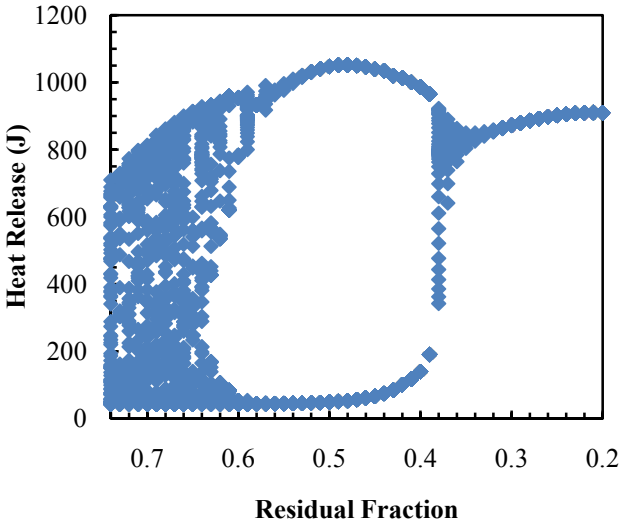

Figure 5.24. Bifurcation of modeled heat release with varying residual fraction for φ=0.7 with no parametric noise

While the behavior with respect to residual fraction is shifted substantially, this behavior qualitatively agrees with past experiments performed by Wagner (75). This form of the model is also able to generate return maps for individual operating conditions that closely resemble those observed experimentally. This is critical for its utility when testing engine controllers in simulations. Even if the input parameters are somewhat shifted from their true values, the cycle-to-cycle dynamics must be similar in order to effectively predict the changes in engine behavior as input parameters are perturbed by a controller. As was shown in section 5.2, this model does show the correct trends when either equivalence ratio or spark timing are varied. Here, the similarity of the dynamics will be shown.

5.3.5. Experimental return maps. The dynamics of the cyclic variability observed in lean and high EGR engine operation have been characterized in detail by Wagner (75), Wagner, et al. (29), and Sutton and Drallmeier (76). An overview of some relevant features is presented here. Return map plots are useful for illustrating the dynamics at an individual operating condition. These graphs plot an event in a sequence versus the prior event in the same sequence. In this case, the heat release for an engine cycle is plotted versus the heat release of the preceding engine cycle. For stochastic variations, time symmetry would be expected, with points falling along the 45° diagonal, as seen in Figure 5.25. The circular shape is typical of normally distributed variations.

For leaner equivalence ratios, time asymmetry begins to be exhibited, with small "arms" spreading out from the central cluster. This progression can be seen in Figure 5.26Figure 5.30. Initially, as seen in Figure 5.26, a few outlying cycles spread to the left and straight down from the central cluster, away from the diagonal, indicating the beginnings of time-asymmetrical behavior.

As dilution is further increased, these outlying cycles then develop into "arms" that give a characteristic "boomerang" shape as in Figure 5.27 and Figure 5.28. The slope of the upper branch is noticeably negative, showing cycles with lower energy combustion events being followed by cycles with higher energy combustion events.

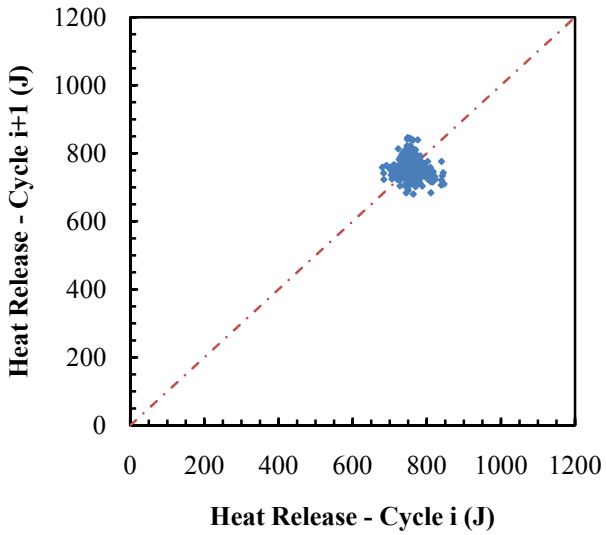

Figure 5.25. Heat release return map for Ricardo engine with φ=0.8, base timing

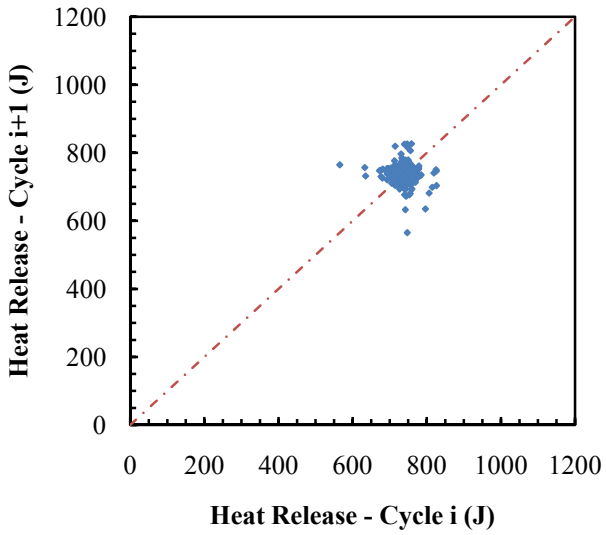

Figure 5.26. Heat release return map for Ricardo engine with φ=0.775, base timing

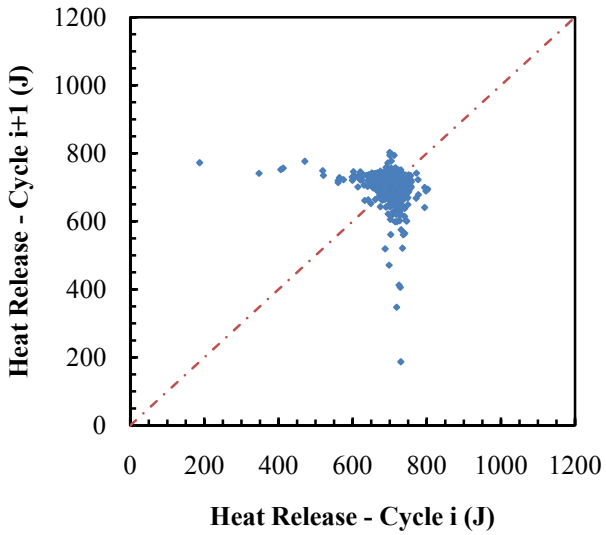

Figure 5.27. Heat release return map for Ricardo engine with φ=0.75, base timing

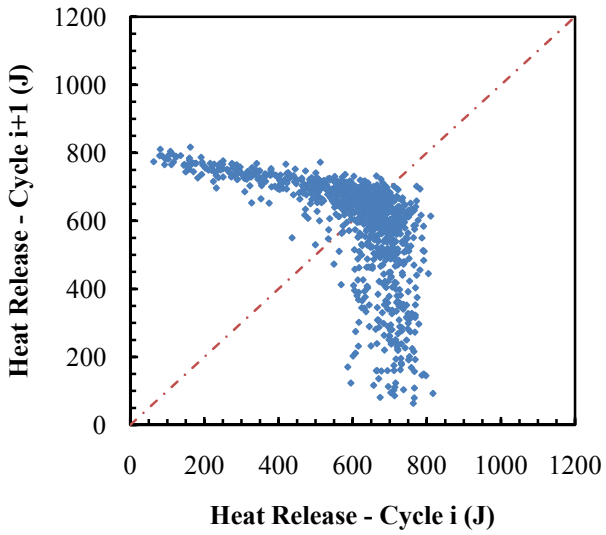

Figure 5.28. Heat release return map for Ricardo engine with φ=0.725, base timing

Continuing to reduce the equivalence ratio past this point results in the majority of cycles being grouped at fixed points towards the upper left and lower right corner, rather than at the center, as seen in Figure 5.29. This is the bifurcated behavior mentioned previously, with alternating high-energy and low-energy combustion events.

Further dilution causes a shift downward towards the origin, with repeated misfires occurring rather than repeated good-quality combustion events, when the dominant high-low pattern is deviated from, as seen in Figure 5.30. Reducing the equivalence ratio even further would result in a concentration of cycles near the origin with occasional outliers up and to the right, then purely motored operation as the lean ignition limit is reached.

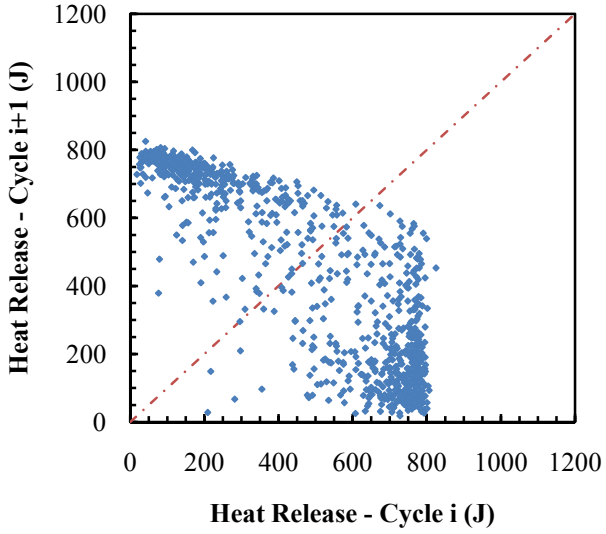

Figure 5.29. Heat release return map for Ricardo engine with φ=0.7, base timing

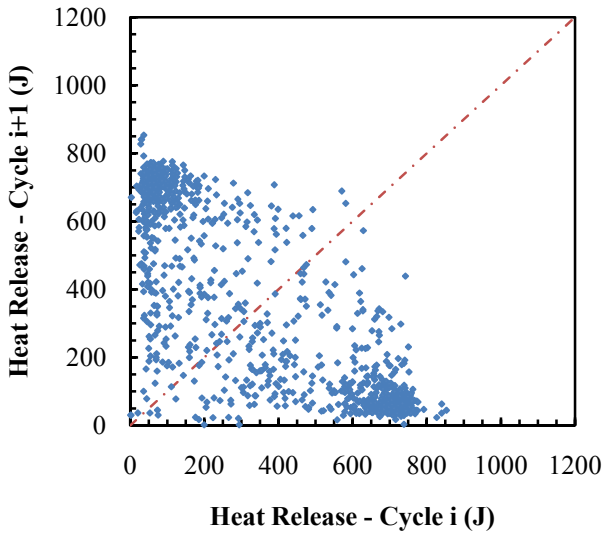

Figure 5.30. Heat release return map for Ricardo engine with φ=0.675, base timing

5.3.6. Model return maps. The same dynamics are observed in the output of the model with turbulent entrainment implemented. As was seen previously with the bifurcation sequences, the behavior is shifted with respect to both residual fraction and equivalence ratio, but the qualitative change in dynamics is the important factor for control. For an equivalence ratio of 0.725, the return map is shown in Figure 5.31. This case, which has very little variation, and no time asymmetry, corresponds to the experimental results for an equivalence ratio of 0.8 in Figure 5.25.

As the level of dilution is increased (by decreasing the equivalence ratio) from this point, the same type of spread, and the same "boomerang-shaped" plots are observed. Figure 5.32 shows the slight extension of "arms" out from the central cluster that was noted in Figure 5.26 for experimental data, albeit at an equivalence ratio of 0.715 rather than 0.775. The negative slope of the upper "arm" is already well-defined in this case.

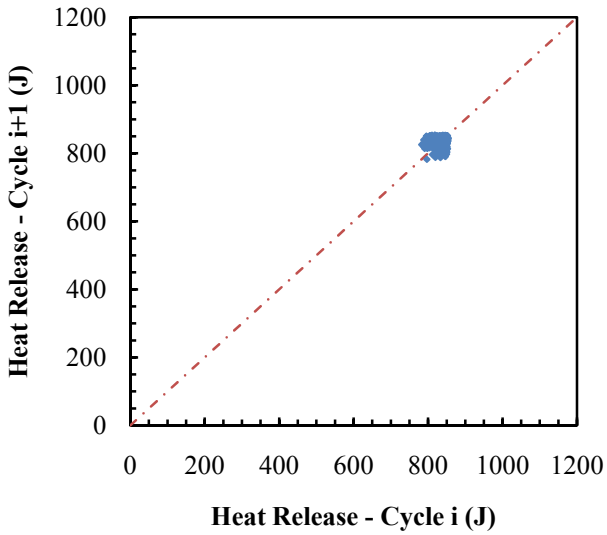

Figure 5.31. Heat release return map for model with φ=0.725

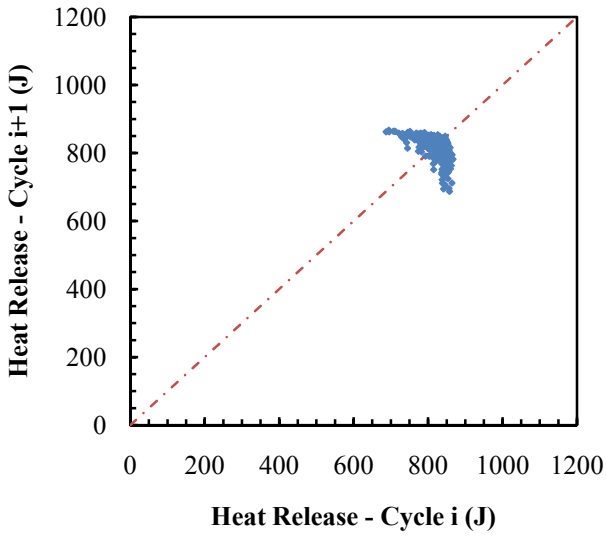

Figure 5.32. Heat release return map for model with φ=0.715

As the equivalence ratio is reduced further, these “arms” grow in the same way, towards the corners of the map, as shown in Figure 5.33. The dynamics here are much the same as for the experimental data at an equivalence ratio of 0.75, shown in Figure 5.27. Further dilution causes further spread along the same pattern, as seen in Figure 5.34.

As with the experimental data, as the equivalence ratio is even further reduced, events will move towards fixed points at the corners of the plot, indicating alternating low-high sequences in Figure 5.35, followed by gravitation to the origin in Figure 5.36.

The good match of the return maps for the model simulations to those produced experimentally makes it preferable to the earlier implementation that lacked the turbulent entrainment model, even though the bifurcation sequences are shifted with respect to equivalence ratio and residual fraction. This prediction of the correct dynamics for high dilution levels, when combined with the prediction of the correct trends as both spark timing and equivalence ratio are varied, fits the requirements for controller development and simulation.

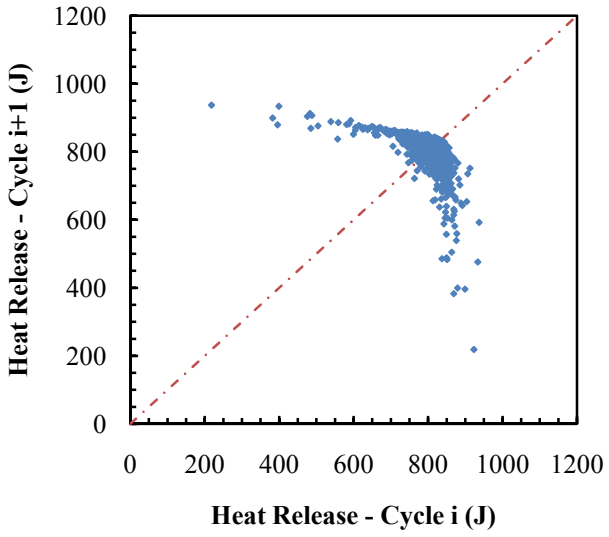

Figure 5.33. Heat release return map for model with φ=0.705

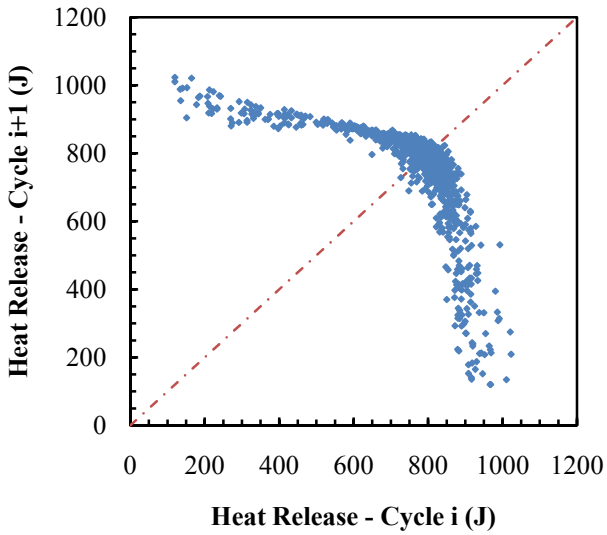

Figure 5.34. Heat release return map for model with φ=0.695

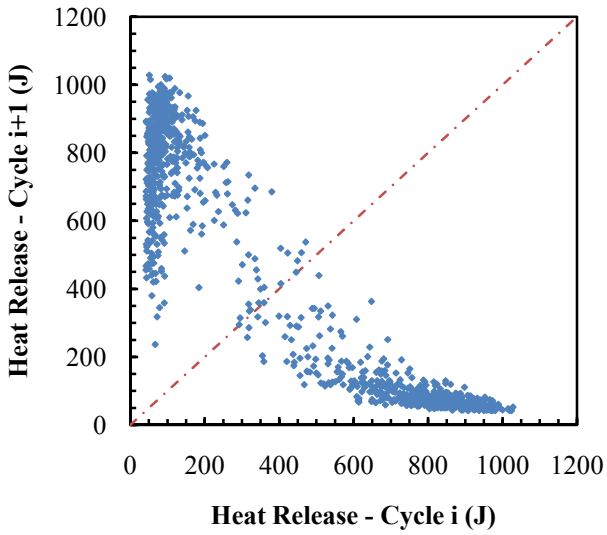

Figure 5.35. Heat release return map for model with φ=0.645

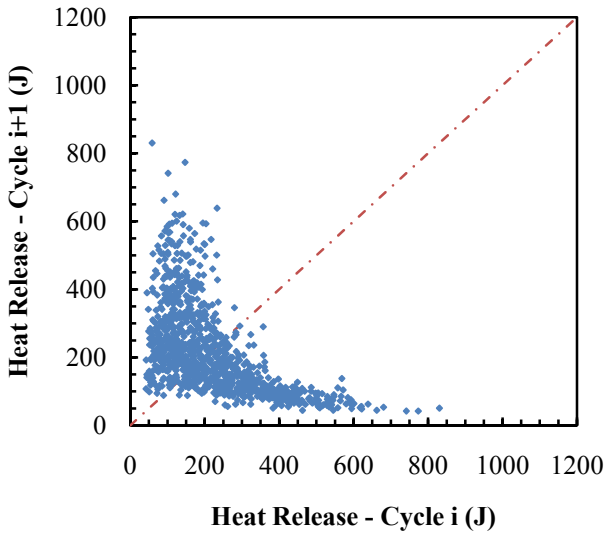

Figure 5.36. Heat release return map for model with φ=0.625

5.3.7. Dilution with EGR. In addition to lean operation, it is also desirable to account for the effects of inert dilution through EGR. The turbulent burning rate equations used include an adjustment for the effect of diluent on laminar flame speed, so this approach was tested to determine whether the same dynamics are observed. Unfortunately, except at very high residual fractions, no bifurcation occurred in the output of the model with respect to EGR. For sufficiently high residual fractions, on the order of 60%, it was possible to produce bifurcated behavior, but the return maps produced are not qualitatively correct. While the characteristic "boomerang" shape observed for lean behavior should be present for inert dilution as well, but instead the model output transitions from the regular, non-bifurcated behavior to a map such as that seen in Figure 5.37. This does not share the key features of observed experimental return maps, as in Figure 5.38, which are very similar to those for lean operation.

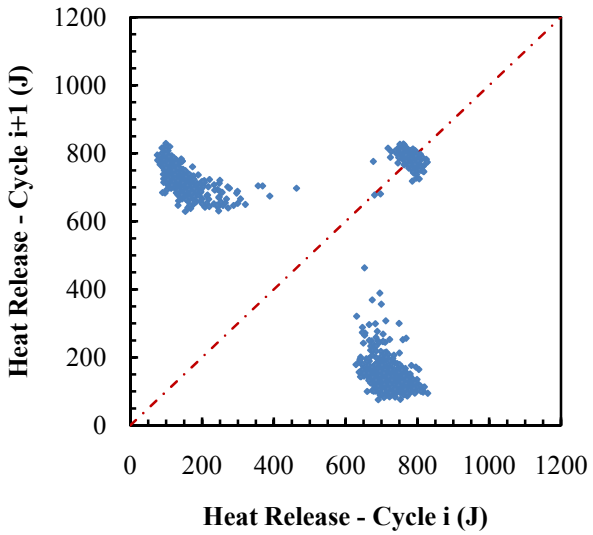

Figure 5.37. Heat release return map for model with EGR dilution

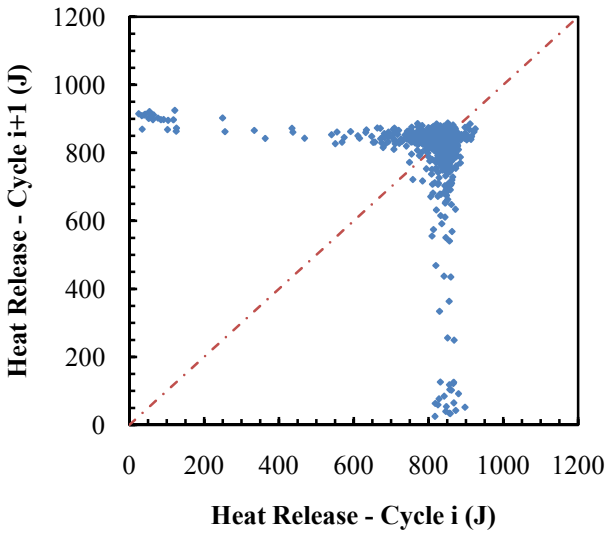

Figure 5.38. Heat release return map for Ricardo engine with EGR dilution

Further investigation into the reason for this inadequacy of the current form of the model for considering high EGR operation is warranted. It appears that the correlation in Equation 5.23 will need to be updated for the dilution levels of interest. While experiments show that dilution through EGR and dilution through excess air cause very similar cycle-to-cycle dynamics (25; 76), the form of the adjustment to the flame speed in the correlations commonly used is very different. The effect of equivalence ratio in these correlations enters by way of an exponential factor on the pressure and temperature ratios, while the effect of inert diluent is merely multiplicative. With the substantially different functional form, it is unsurprising that the output is qualitatively different as EGR level is varied in the model. Better correlations will therefore be required to account for the effects of high levels of inert diluent on cyclic variations in SI engine operation.

In the meantime, however, a method like that used to modify the model of Daw, et al. (26) to account for EGR can be used. Since the dynamics of lean and high EGR operation are qualitatively very similar, an effective equivalence ratio that considers the masses of air and fuel and also includes the inert diluent as if it were excess air can be determined for each cycle and input into the burning rate model. In this manner, an equivalent lean operation condition will be simulated, producing a qualitatively correct return map for use in simulating controller performance.

5.4. CONCLUSIONS

A two-zone thermodynamic model was combined with a turbulent mass entrainment combustion model to simulate the cycle-to-cycle dynamics of highly dilute SI operation. The increased physical detail of this model over those previously used for cycle-to-cycle studies allows for greater insight into the variation of state variables as the cycle progresses, and allows the model to predict the correct trends as not only composition, but also spark timing and other such parameters are varied. This model also successfully produces qualitatively correct return maps and bifurcation sequences, which is important for its utility in control development and simulation.

Some further improvement is desirable, since the output of the model is shifted with respect to residual fraction, when compared to experimental results. Also, the form of the correlation used to account for inert dilution through EGR is not adequate to

capture the observed dynamics. The nonlinearities in flame speed that are present for lean equivalence ratios are not duplicated for inert diluents. These shortcomings can be worked around for the present, but constitute fertile ground for future development efforts.

6. SUMMARY AND CONCLUSIONS

6.1. SUMMARY

Several topics were researched that will address outstanding issues with the control of cycle-to-cycle variations in dilute SI combustion. First, a novel application of nonlinear dynamics analysis methods allows for prediction of the effectiveness of control at various operating conditions, which will aid in determining when certain combustion modes would best be utilized. Second, experiments were performed to determine the sensitivity of an engine operating under dilute conditions to perturbations of control inputs. This information is necessary in order to evaluate whether a controller being developed is giving reasonable outputs. Finally, a turbulent mass entrainment combustion model was combined with a thermodynamic model in such a manner as to enable multi-cycle simulations of engine operation that is based on real parameters such as equivalence ratio and spark timing, rather than arbitrary mathematical constants.

Symbol sequence analysis of experimental data provides a measure of the determinism present. This method was used here to identify the individual engine cycles that contributed to deterministic variations, and remove them from a "cleaned" set of data. These modified, clean data sets contain the remaining cycles, and the variations that occur within them are those that are due to stochastic effects, rather than determinism. Analysis of these cleaned data illustrates the improvement that could be realized with a controller that is effective at eliminating the deterministic variations in engine output, and further gives insight into when highly dilute SI operation would be beneficial as opposed to other combustion modes.

In the development of such controllers, the question also arises when evaluating a controller's performance of whether the output of the controller is reasonable in magnitude. Experiments were conducted on an SI engine under dilute operating conditions, wherein the fuel pulse width was varied in a sinusoidal manner. An FFT of the engine heat release data was examined to determine whether the frequency corresponding to the imposed variation in the control input was significant compared to variations at other frequencies. The results indicate that while greater changes are needed

for stoichiometric operation, perturbations on the order of 1% were easily detected for higher levels of dilution.

The multi-cycle engine model presented combines a two-zone thermodynamic model with a turbulent mass-entrainment model, and carries over residual gas composition and temperature from cycle to cycle. It successfully simulates the same trends seen in experimental data with regards to changes in equivalence ratio, spark timing, and other parameters, with input parameters being restricted to actual physical parameters of the system rather than arbitrary mathematical constants to tune the model behavior. The added physical detail compared to simpler models allows for examination of internal state variables in different cycles. While the cyclic dynamics are shifted with respect to the values of some input parameters, the qualitative match is very good, and the trends are correctly predicted as those parameters are changed. This will enable simulation of controllers that use spark timing or other inputs than fuel pulse width.

These contributions should be beneficial to future development of controllers to address the dynamical cycle-to-cycle variations in dilute SI engine behavior, and in determining what operating conditions are appropriate for this combustion mode in general.

6.2. SUGGESTED FUTURE WORK

The results here suggest several areas of future work that would be desirable. With respect to the examination of different control inputs, two have already been mentioned. It would be desirable to evaluate the sensitivity of the engine to changes in both spark timing and EGR level as control inputs. Spark timing will be the easier of these parameters to test: once an appropriate operational amplifier is added to the output of the control hardware, such an experiment can be performed with otherwise existing hardware and software.

In order to examine the sensitivity of the engine to perturbations of EGR as a control input, additional engine control hardware must be developed. The easiest approach to implement using simulated EGR from bottled nitrogen would be to use a solenoid valve such as a fuel injector to add a small supplemental charge of diluent for each engine cycle to the bulk, nominal mass that is present for every cycle. A more complex approach that has the advantage of using true exhaust gases for EGR rather than

simulating the effects with bottled gas would be to implement an electronically controlled VVT system on the engine. With such a system, the residual gas could be changed from cycle to cycle by modifying the valve timing. If the engine is found to be sufficiently sensitive to changes in diluent, this approach would offer the advantage of retaining a fixed, stoichiometric air/fuel ratio that could be used with current three-way catalytic converters for emission control.

Further development of the multi-cycle engine model is also warranted. In particular, an ignition module would be necessary to predict complete misfires, and a more advanced heat transfer model would improve the accuracy of the thermodynamic model, especially towards the end of the engine cycle where the current model deviates significantly from experimental data. These improvements may or may not be sufficient to compensate for the shift in the location of dynamical transitions with respect to parameters such as equivalence ratio and residual fraction; if not, further development should be aim to improve the prediction in that manner as well. Better correlation parameters for flame speeds at high dilution levels may be required in such a case. Additionally, better correlations to account for the effects of high levels of inert diluent should be developed to correct the model's EGR predictions.

Also, further "computational experiments" with the model, either in the current form or an improved one, could give added insight into the effects of changing various parameters of interest that are difficult to measure experimentally. While the results do not yet perfectly align with experimental data, the dynamics and trends are correctly predicted, so valuable insights could be gained from such examinations, possibly leading to new, more effective control strategies. Not only could the effect of perturbing various parameters in a controller be tested, but considerations such as the choice of fuel could also be examined.

APPENDIX A
SYMBOLIC ANALYSIS SOURCE CODE

The Fortran 95 source code for generating symbol sequence histograms and cleaning the data is contained in this appendix, and can also be found on the CD accompanying this dissertation. Syntax help for running the program can be viewed by running the program with the help command: `secompare.exe /?`

A brief description of the modules, subroutines, and functions found in each source file follows:

```
SECompare.f90
    Program SECompare               Main program
        Uses:                       ISO_Varying_String
                                    CharConvert
                                    F2kCLI
    Subroutine Cutoff               Determine boundaries for partitions
        Arguments (in):             npart, npts
        Arguments (in/out):         ax
        Arguments (out):            axcut
    Subroutine HeapSort             Sorts data in ascending order
        Arguments (in):             k
        Arguments (in/out):         ra
    Subroutine SECHelp              Displays syntax help
BKFileIO.f90
    Subroutine BKInFile             Opens an existing file for input
        Arguments (in/out):         FileIn
        Arguments (out):            funit
    Subroutine BKOutFile            Creates a new file for output
        Arguments (in/out):         FileOut
        Arguments (out):            funit
    Subroutine BKFClose             Closes specified files
        Arguments (in):             n, funit
    Subroutine BKIOErr              Returns text for Fortran I/O error status (NAS Compiler)
        Uses:                       FORTRAN_IO_ERRORS
        Arguments (in):             ios
char_conv.f90
    Module CharConvert
        Function CharToInt          Converts string to integer
            Arguments (in):         InString
        Function CharToReal         Converts string to real number
            Arguments (in):         InString
        Function CharToDouble       Converts string to double precision number
            Arguments (in):         InString
        Function CharToComplex      Converts string to complex number
            Arguments (in):         InString
        Function IntToChar          Converts integer to string
            Arguments (in):         InNum
        Function RealToChar         Converts real number to string
            Arguments (in):         InNum
```

```
*** BEGIN FILE SECOMPARE.F90 ***

!   Compare Symbol Sequence Histograms for engine data with theoretical "best case" where
!   alternating sequences are removed.

!   Brian Kaul, 2004

!   Portions of the symbolic nonlinear analysis have been translated to Fortran from
!   Robert Wagner's BASIC codes in the Appendices of his PhD Dissertation.

Program SECompare
!   Use generic_sort
    Use F2kCLI
    Use CharConvert, Only CharToInt,IntToChar,CharToReal
    Use ISO_Varying_String

    Implicit None

    Integer :: i,j,k,l,nargs,n,npart,nbins,nseq,seqlength,ios,ndata,cnt,newsets,stdtype
    Integer, Parameter :: numfiles=4
    Integer, Dimension(numfiles) :: funit
    Real, Dimension(:), Allocatable :: q,Freq,qs,qb,qbin,Seq,SeqSort,FreqSeq,qcut,IMEPn,IMEPg
    Real :: SEntropy,EntropyMin,Idx,power,FreqMean,FreqStDev,stdevs,noisebase
    Character(Len=100), Dimension(numfiles) :: fname
    Character(Len=100) :: InTemp,dummy1,dummy2,dummy3,dummy4,dummy5
    Character(Len=7) :: fnum1,fnum2,fnum3
    Character(Len=1) :: basis
    Logical, Dimension(:), Allocatable :: HighCount,KillSeq,Killq
    Logical :: FCon

    nargs = Command_Argument_Count()
    fname = "data.csv"
    n = 1000
    npart = 2
    seqlength = 6
    stdevs = 1
    stdtype = 1
    basis = 'q'
    If (nargs > 0) Then
        Do i=1,nargs
            Call Get_Command_Argument(number=i,value=InTemp)
            Select Case (i)
                Case (1)
                    If ((Index(InTemp,'-',back=.false.) == 1).or.(Index(InTemp,'/',back=.false.)
== 1)) Then
!                       Select Case (Extract(Intemp,2,2))    ! Display help and exit
!                           Case ('h','H','?')
                                Call SECHelp()
                                Stop
!                           Case Default
!                               Stop 'Filenames beginning with "-" or "/" are not allowed.'
!                       EndSelect
                    Else
                        fname(i) = InTemp
                        If (.not. (Index(fname(1),'.',back=.true.) > 1)) Then
                            fname(i) = fname(i) // '.csv'
                        EndIf
                    EndIf
                Case (2)
                    n = CharToInt(InTemp)
                Case (3)
                    npart = CharToInt(InTemp)
                Case (4)
                    seqlength = CharToInt(InTemp)
                Case (5)
                    stdevs = CharToReal(InTemp)
                Case (6)
                    stdtype = CharToInt(InTemp)
                Case (7)
```

```
                    basis = InTemp
                Case Default
                    Write (unit=*,fmt=100) 'Too many arguments detected.  Additional arguments
ignored; behavior may not be as expected.'
                    Exit
            EndSelect
        EndDo
    Else
        Call SECHelp()
        Stop
    EndIf
    Allocate(q(n),qs(n),qb(n),qbin(n),Freq(n),Seq(n),SeqSort(n),FreqSeq(n),KillSeq(n),Killq(n),IME
Pn(n),IMEPg(n))
    Allocate(qcut(npart))
    noisebase = (1/Real(npart))**seqlength

    fnum1 = IntToChar(npart)
    fnum1 = Extract(fnum1,Index(Trim(fnum1),' ',back=.true.)+1,Len(Trim(fnum1)))
    fnum2 = IntToChar(seqlength)
    fnum2 = Extract(fnum2,Index(Trim(fnum2),' ',back=.true.)+1,Len(Trim(fnum2)))
    fname(2) = Extract(fname(1),1,Index(fname(1),'.',back=.true.)-1) // '-' // Trim(fnum1) // '-'
// Trim(fnum2) // '-symb.csv'
    fname(3) = Extract(fname(1),1,Index(fname(1),'.',back=.true.)-1) // '-' // Trim(fnum1) // '-'
// Trim(fnum2) // '-clean'
    fname(4) = Trim(fname(3)) // '.csv'

    Call BKInFile(fname(1),funit(1))
    Call BKOutFile(fname(2),funit(2))

!   Get data from input file
    Read (unit=funit(1),fmt=*,iostat=ios) dummy1,dummy2,dummy3,dummy4,dummy5
    Do i=1,n
        Read (unit=funit(1),fmt=*,iostat=ios) j,IMEPn(i),IMEPg(i),q(i),dummy1
        If (ios < 0) Then
            If (ndata < seqlength) Write (unit=*,fmt=100) 'Number of data points is less than
sequence length.'
            Exit
        ElseIf (ios > 0) Then
            Call BKIOErr(ios)
        EndIf
        ndata=ndata+1
    EndDo

    Select Case (basis)
        Case ('i','I')
            qb = IMEPn
        Case ('q','Q')
            qb = q
        Case Default
            Write (unit=*,fmt=100) 'Invalid basis argument.  Basis must be "i" or "q".'
            Stop
    EndSelect

!   ____________________________________
!   Begin section adapted from Wagner's code

!   Subroutine Symbol. Subroutine converts original data sequence to symbol sequence, and
calculates
!   modified Shannon entropy and symbol sequence histogram based on user specified number
!   of partitions and sequence length. Subroutine is based on a program written by C. S. Daw, Oak
!   Ridge National Laboratory, 1996.

    EntropyMin = 1
!   determine cutoff values for symbol conversion
    qs = qb
    Call Cutoff(npart,n,qs,qcut)
!   symbol conversion
    Do i=1,n
        qbin(i) = 0
```

```
        Do j=1,npart-1
            If (qb(i) >= qcut(j)) qbin(i) = j
        EndDo
    EndDo
!   number of different sequences and sequences in original data series
    nbins = npart**seqlength
    nseq = n-seqlength+1
!   determine sequence equivalents
    Do i=1,nseq
        Idx = 0
        Do j=0,seqlength-1
            power = seqlength-1-j
            Idx = Idx+qbin(i+j)*npart**power
        EndDo
        Seq(i) = Idx
    EndDo
    SeqSort = Seq
!   sort sequence equivalents in preparation for binning
    Call HeapSort(nseq,SeqSort)
!   determine frequency of nonzero sequences
    cnt = 1
    Freq(1) = 1
    FreqSeq(1) = SeqSort(1)
    Do i=2,nseq
        If (SeqSort(i) /= SeqSort(i-1)) Then
            cnt = cnt+1
            FreqSeq(cnt) = SeqSort(i)
        EndIf
        Freq(cnt) = Freq(cnt)+1
    EndDo
    Do i=1,cnt
        Freq(i) = Freq(i)/Real(nseq)
        SEntropy = SEntropy-Freq(i)*Log(Freq(i))
    EndDo
    If (cnt > 1) Then
        SEntropy = SEntropy/Log(cnt)
    EndIf

!   End section adapted from Wagner's code
!   ______________________________________

    Write (unit=funit(2),fmt=115)
    Do i=1,cnt
        Write (unit=funit(2),fmt=120) FreqSeq(i),Freq(i)
    EndDo
    Write (unit=funit(2),fmt=130) SEntropy

    Allocate(HighCount(cnt))
    HighCount = .false.

    FreqMean = Sum(Freq)/Real(cnt)
    FreqStDev = Sqrt(Sum((Freq(1:cnt)-FreqMean)**2)/Real(cnt-1))

!   Locate which sequences have high peaks on histogram
    Do i=1,cnt
        If (stdtype == 1) Then
            If (Freq(i) >= FreqMean+stdevs*FreqStDev) HighCount(i) = .true.
        Else
            If (Freq(i) >= noisebase*stdevs) HighCount(i) = .true.
        EndIf
    EndDo
!   Determine which sequences (not sorted) correspond to high peaks - to remove from "cleaned"
output
!   Neglect peaks corresponding to max value (i.e. all strong cycles)
    KillSeq = .false.
    Do i=1,nseq
        Do j=1,cnt
            If (FreqSeq(j) == Seq(i)) Then
                If (Seq(i) /= Maxval(Seq)) Then
```

```
                    If (HighCount(j)) KillSeq(i) = .true.
                EndIf
                Exit
            EndIf
        EndDo
    EndDo
!   Determine which data points are in high sequences
    Killq = .false.
    newsets = 1

    fname(3) = Trim(fname(3)) // '-1.csv'
    Call BKOutFile(fname(3),funit(3))
    Call BKOutFile(fname(4),funit(4))
    Write (unit=funit(3),fmt=105,iostat=ios)
    Do i=2,n
        Do j=0,seqlength-1
            if (i-j < 1) Exit
            If (KillSeq(i-j)) Killq(i) = .true.
        EndDo
        If (.not. Killq(i-1)) Then
            Write (unit=funit(3),fmt=110,iostat=ios) i-1,q(i-1),IMEPn(i-1),IMEPg(i-1)
            If (.not. Killq(i)) Then
                Write (unit=funit(4),fmt=111,iostat=ios) i-1,q(i-1),IMEPn(i-1),IMEPg(i-1),q(i)
            Else
                Write (unit=funit(4),fmt=110,iostat=ios) i-1,q(i-1),IMEPn(i-1),IMEPg(i-1)
            EndIf
            If (ios /= 0) Call BKIOErr(ios)
        ElseIf (Killq(i) /= Killq(i-1)) Then         ! Start next "new set" from filtered data
            Call BKFClose(1,funit(3))
            newsets = newsets+1
            fnum3 = IntToChar(newsets)
            fnum3 = Extract(fnum3,Index(Trim(fnum3),' ',back=.true.)+1,Len(Trim(fnum3)))
            fname(3) = Extract(fname(3),1,Index(fname(3),'-',back=.true.)) // Trim(fnum3) //
'.csv'
            Call BKOutFile(fname(3),funit(3))
            Write (unit=funit(3),fmt=105,iostat=ios)
            If (ios /= 0) Call BKIOErr(ios)
        EndIf
    EndDo
    If (.not. Killq(n-1)) Then
        If (.not. Killq(n)) Then
            Write (unit=funit(3),fmt=111,iostat=ios) n-1,q(n-1),IMEPn(n),IMEPg(n),q(n)
        Else
            Write (unit=funit(3),fmt=110,iostat=ios) n-1,q(n-1),IMEPn(n),IMEPg(n)
        EndIf
    EndIf

    Call BKFClose(4,funit)

    Write (unit=*,fmt=100) 'Finished'

    100 Format (a)
    105 Format ('Cycle,Heat Release,IMEP Net,IMEP Gross,HR+1')
    110 Format (i5,3(',',E15.8E3))
    111 Format (i5,4(',',E15.3E3))
    115 Format ('Sequence,Frequency')
    120 Format (F15.5,',',F8.5)
    130 Format ('Shannon Entropy,',E15.8E3)
    150 Format (i5,:5(',',a)) !  fix format and test

End Program SECompare

Subroutine Cutoff(npart,npts,ax,axcut)
!   Translated to Fortran from Robert Wagner's BASIC codes in his PhD Dissertation.  Comments
!   are mostly original, and refer to sections of his Dissertation.

!   Subroutine CutOff. Subroutine determines location of partition boundaries corresponding to
!   equiprobable bins for converting the original time series to a symbol sequence. Refer to
Section
```

```
!   4.2.3 for a discussion of data partitioning.
    Implicit None

    Integer, Intent(in) :: npart,npts
    Real, Dimension(npts), Intent(inout), Target :: ax
    Real, Dimension(npart), Intent(out) :: axcut

    Integer :: j,upper,lower
    Real :: fract

!   sort values
    Call HeapSort(npts,ax)
!   determine cutoff values
    Do j=1,npart-1
        fract = Real(j/Real(npart))*npts+0.5
        upper = Int(fract+1)
        lower = Int(fract)
        axcut(j) = (ax(upper)-ax(lower))*(fract-lower)+ax(lower)
    EndDo
    Return
End Subroutine Cutoff

Subroutine HeapSort(k,ra)
!   Translated to Fortran from Robert Wagner's BASIC codes in his PhD Dissertation.

!   Subroutine HeapSort. Subroutine sorts data in ascending order and is used with the subroutine
!   CutOff to determine partition boundaries.
Implicit None

    Integer, Intent(in) :: k
    Real, Dimension(k), Intent(inout) :: ra

    Integer :: i,j,l,ir
    Real :: RRA

    l = Int(0.5*k)+1
    ir = k
    Do
        If (l > 1) Then
            l = l-1
            RRA = ra(l)
        Else
            RRA = ra(ir)
            ra(ir) = ra(1)
            ir = ir-1
            If (ir == 1) Then
                ra(1) = RRA
                Return
            EndIf
        EndIf
        i = l
        j = l+l
        Do While (j <= ir)
            If ((j < ir).and.(ra(j) < ra(j+1))) j=j+1
            If (RRA < ra(j)) Then
                ra(i) = ra(j)
                i = j
                j = j+j
            Else
                j = ir+1
            EndIf
        EndDo
        ra(i) = RRA
    EndDo
    Return
End Subroutine HeapSort

Subroutine SECHelp()
    Implicit None
```

```
    Write (unit=*,fmt=100) 'Calculates Shannon Entropy and symbol frequency for original data and
generates'
    Write (unit=*,fmt=100) 'modified data with oscillations removed.  Calculates COV for both
sets. Output'
    Write (unit=*,fmt=100) 'files will be automatically named based on the input file andsettings,
and will'
    Write (unit=*,fmt=100) 'be in CSV format.  If input filename is given without extension, the
extension'
    Write (unit=*,fmt=100) 'will be assumed to be .csv - if it is not, the file will not be
found.'
    Write (unit=*,fmt=100)
    Write (unit=*,fmt=100) ' Syntax:'
    Write (unit=*,fmt=100) ' SECompare <input filename> <n> <npart> <seqlngth> <stdevs> <stdtype>
<basis> [-help]'
    Write (unit=*,fmt=100)
    Write (unit=*,fmt=100) ' n          Number of data points to consider (total)'
    Write (unit=*,fmt=100) ' npart      Number of partitions to use'
    Write (unit=*,fmt=100) ' seqlngth   Length of sequence to consider'
    Write (unit=*,fmt=100) ' stdevs     Number of standard deviations away from average frequency'
    Write (unit=*,fmt=100) '            for clean data cutoff'
    Write (unit=*,fmt=100) ' stdtype    Use standard deviations for cutoff if 1, else stdevs is'
    Write (unit=*,fmt=100) '            used as a multiplier on the basic noise level'
    Write (unit=*,fmt=100) ' basis      If "i" IMEP is used as a basis; if "q" Heat Release is
used'
    Write (unit=*,fmt=100)
    Write (unit=*,fmt=100) ' -help      Displays this help message'
    Write (unit=*,fmt=100) '              (also: -?, /?, /help, or no argument)'

    100 Format (a)

End Subroutine SECHelp

!   EOF

*** END FILE SECOMPARE.F90 ***
```

```
*** BEGIN FILE BKFILEIO.F90 ***

Subroutine BKInfile(FileIn,funit)
!   Prompt for file name and open file

!   FileIn = Input File Name; FExist = File Exists - logical
!   FChoice = File replacement menu choice; FSuccess = File operation successful
!   i = loop counter; ios = IO Status indicator

    Implicit None
    Integer, Intent(out) :: funit
    Logical :: FExist,FSuccess,FCon
    Character(len=*), Intent(inout) :: FileIn
    Integer :: i,ios

    FSuccess = .false.
    Do While (.not. FSuccess)
        Inquire (file=FileIn,exist=FExist)
        Do i=10,99
            Inquire (unit=i,opened=FCon)
            If (.not. FCon) Then
                funit=i
                Exit
            EndIf
        EndDo
        If (FExist) Then
            Open (unit=funit,file=FileIn,status='old',iostat=ios,action='read',position='rewind')
            If (ios.ne.0) Then
                Call BKIOErr(ios)
            Else
                FSuccess = .true.
            EndIf
        Else
            Write (unit=*,fmt=100) 'Input File not found; Enter new filename:'
            Read (unit=*,fmt='(a)') FileIn
        EndIf
    EndDo
    Return
    100 Format (a)
End Subroutine BKInfile

Subroutine BKOutfile(FileOut,funit)
!   FileOut = Output File Name; FExist = File Exists - logical
!   FChoice = File replacement menu choice; FSuccess = File operation successful
!   i = loop counter; ios = IO Status indicator

    Implicit None
    Integer, Intent(out) :: funit
    Logical :: FExist,FSuccess,FCon
    Character(len=*), Intent(inout) :: FileOut
    Character(len=1) :: FChoice
    Integer :: i,ios

    FSuccess = .false.
    Do While (.not. FSuccess)
        Inquire (file=FileOut,exist=FExist)
        Do i=10,99
            Inquire (unit=i,opened=FCon)
            If (.not. FCon) Then
                funit=i
                Exit
            EndIf
        EndDo
        If (FExist) Then
            Do
                Write (unit=*,fmt=110) Trim(FileOut)
                Write (unit=*,fmt=100) '1. (R)eplace'
                Write (unit=*,fmt=100) '2. (A)ppend'
                Write (unit=*,fmt=100) '3. Enter (N)ew filename'
```

```
                Read (unit=*, fmt='(a)') FChoice
                Select Case (FChoice)
                    Case ('1','R','r')
                        Open
(unit=funit,file=FileOut,status='replace',iostat=ios,action='write',position='rewind')
                        If (ios.ne.0) Then
                            Call BKIOErr(ios)
                        Else
                            FSuccess = .true.
                            Exit
                        EndIf
                    Case ('2','A','a')
                        Open
(unit=funit,file=FileOut,status='old',iostat=ios,action='write',position='append')
                        If (ios /= 0) Then
                            Call BKIOErr(ios)
                        Else
                            FSuccess = .true.
                            Exit
                        EndIf
                    Case ('3','N','n')
                        Write (unit=*,fmt=100) 'Enter new output file name:'
                        Read (unit=*,fmt='(a)') FileOut
                        Exit
                    Case Default
                        Cycle
                EndSelect
            EndDo
        Else
            Open
(unit=funit,file=FileOut,status='new',iostat=ios,action='write',position='rewind')
            If (ios.ne.0) Then
                Call BKIOErr(ios)
            Else
                FSuccess = .true.
            EndIf
        EndIf
    EndDo
    Return
    100 Format (a)
    110 Format ('File ',a,' already exists.')
End Subroutine BKOutfile

Subroutine BKFClose(n,funit)
!   Close input/output file(s)

!   funit = unit of file to close; ioerr = return error variable; ios = IO Status indicator
!   i = loop counter; n = number of files to close

    Implicit None
    Integer, Intent(in) :: n
    Integer, Dimension(n), Intent(in) :: funit
    Integer :: i,ios

    Do i=1,n
        Close (unit=funit(i),iostat=ios)
        If (ios.ne.0) Then
            Call BKIOErr(ios)
        EndIf
    EndDo
    Return
    100 Format (1x,a)
End Subroutine BKFClose

Subroutine BKIOErr(ios)
!   Return Error message for iostat variable ios - for NAS FortranPlus v2 compiler
    Use FORTRAN_IO_ERRORS

    Implicit None
```

```
    Integer, Intent(in) :: ios

    Select Case (ios)
    Case (-2:-1,1:67)
        Write (unit=*,fmt=100) ios,IOERR_MESSAGES(ios)
    Case (0)
        Write (unit=*,fmt=110)
    Case Default
        Write (unit=*,fmt=120)
    EndSelect
    Return

    100 Format ('I/O Error ',i2,' - ',a)
    110 Format ('I/O Operation Completed Successfully.')
    120 Format ('Unknown I/O Error number: ',i2)
End Subroutine BKIOErr

*** END FILE BKFILEIO.F90 ***
```

```
*** BEGIN FILE CHARCONVERT.F90 ***

Module CharConvert
!   Take a string value that represents a numeric value, and read it into a numeric variable
!   There may be more elegant ways than using a scratch file, but this was quick/easy and works.

    Contains
    Function CharToInt(InString)
        Implicit None
        Integer :: CharToInt
        Character(Len=*), Intent(in) :: InString

        Integer i,unitnum
        Logical FCon

        Do i=10,99
            Inquire (unit=i,opened=FCon)
            If (.not. FCon) Then
                unitnum=i
                Exit
            EndIf
        EndDo

        Open
(unit=unitnum,status='scratch',access='sequential',form='formatted',blank='null',action='readwrite
',position='rewind')
        Write (unit=unitnum,fmt='(a)') InString
        Rewind (unit=unitnum)
        Read (unit=unitnum,fmt=*) CharToInt
        Close (unit=unitnum)
        Return
    End Function CharToInt

    Function CharToReal(InString)
        Implicit None
        Real :: CharToReal
        Character(Len=*), Intent(in) :: InString

        Integer i,unitnum
        Logical FCon

        Do i=10,99
            Inquire (unit=i,opened=FCon)
            If (.not. FCon) Then
                unitnum=i
                Exit
            EndIf
        EndDo

        Open
(unit=unitnum,status='scratch',access='sequential',form='formatted',blank='null',action='readwrite
',position='rewind')
        Write (unit=unitnum,fmt='(a)') InString
        Rewind (unit=unitnum)
        Read (unit=unitnum,fmt=*) CharToReal
        Close (unit=unitnum)
        Return
    End Function CharToReal

    Function CharToComplex(InString)
        Implicit None
        Complex :: CharToComplex
        Character(Len=*), Intent(in) :: InString

        Integer i,unitnum
        Logical Fcon

        Do i=10,99
            Inquire (unit=i,opened=FCon)
```

```
            If (.not. FCon) Then
                unitnum=i
                Exit
            EndIf
        EndDo

        Open
(unit=unitnum,status='scratch',access='sequential',form='formatted',blank='null',action='readwrite
',position='rewind')
        Write (unit=unitnum,fmt='(a)') InString
        Rewind (unit=unitnum)
        Read (unit=unitnum,fmt=*) CharToComplex
        Close (unit=unitnum)
        Return
    End Function CharToComplex

    Function CharToDouble(InString)
        Implicit None
        Double Precision :: CharToDouble
        Character(Len=*), Intent(in) :: InString

        Integer i,unitnum
        Logical FCon

        Do i=10,99
            Inquire (unit=i,opened=FCon)
            If (.not. FCon) Then
                unitnum=i
                Exit
            EndIf
        EndDo

        Open
(unit=unitnum,status='scratch',access='sequential',form='formatted',blank='null',action='readwrite
',position='rewind')
        Write (unit=unitnum,fmt='(a)') InString
        Rewind (unit=unitnum)
        Read (unit=unitnum,fmt=*) CharToDouble
        Close (unit=unitnum)
        Return
    End Function CharToDouble

    Function IntToChar(InInt)
        Implicit None
        Integer, Intent(In) :: InInt
        Character(Len=*) :: IntToChar

        Integer i,unitnum
        Logical FCon

        Do i=10,99
            Inquire (unit=i,opened=FCon)
            If (.not. FCon) Then
                unitnum=i
                Exit
            EndIf
        EndDo

        Open
(unit=unitnum,status='scratch',access='sequential',form='formatted',blank='null',action='readwrite
',position='rewind')
        Write (unit=unitnum,fmt=*) InInt
        Rewind (unit=unitnum)
        Read (unit=unitnum,fmt='(a)') IntToChar
        Close (unit=unitnum)
        Return
    End Function IntToChar

    Function RealToChar(InReal)
```

```
        Implicit None
        Real, Intent(In) :: InReal
        Character(Len=*) :: RealToChar

        Integer i,unitnum
        Logical FCon

        Do i=10,99
            Inquire (unit=i,opened=FCon)
            If (.not. FCon) Then
                unitnum=i
                Exit
            EndIf
        EndDo

        Open
(unit=unitnum,status='scratch',access='sequential',form='formatted',blank='null',action='readwrite
',position='rewind')
        Write (unit=unitnum,fmt=*) InReal
        Rewind (unit=unitnum)
        Read (unit=unitnum,fmt='(a)') RealToChar
        Close (unit=unitnum)
        Return
    End Function RealToChar

End Module CharConvert

*** END FILE CHARCONVERT.F90 ***
```

APPENDIX B
FFT ANALYSIS MATLAB SCRIPT

The MATLAB script for calculating the FFT of heat release sequences and plotting the power content is contained in this appendix, and can also be found on the CD accompanying this dissertation.

```
%**************************************************************************
% Brian Kaul
% hrfft.m
% Analyze Engine Control Sensitivity Data
%**************************************************************************

clear all; close all; clc;

%**************************************************************************

fntmp=input('Enter XXX of egrXXX-pYY-aZZZZ-###-h.csv type file: ','s');
%filename=['egr' fntmp '-p'];
filename=['egr' fntmp '-p50-a'];
%fntmp=input('Enter YY of egrXXX-pYY-aZZZZ-###-h.csv type file: ','s');
%filename=[filename fntmp '-a'];
fntmp=input('Enter ZZZZ of egrXXX-pYY-aZZZZ-###-h.csv type file: ','s');
%filename=[filename fntmp '-'];
filename=[filename fntmp '-001-h.csv'];
%fntmp=input('Enter ### of egrXXX-pYY-aZZZZ-###-h.csv type file: ','s');
%filename=[filename fntmp '-h.csv'];
funit=fopen(filename,'r');
[Header] = textscan(funit,'%s %s %s %s %s',1,'delimiter',',');
[Data] = textscan(funit,'%d %f %f %f %f','delimiter',',');
fclose(funit);
sizec=size(Data);
Data=Data(:,1:sizec(2)-1);
k=Data(1);
IMEPn=Data(2);
IMEPg=Data(3);
HR=Data(4);
q=HR{1,1};
%HR2=Data(5);

%**************************************************************************

y=fft(q);
m=size(y);
n=m(1);
Y=y.*conj(y)/n;

x=transpose(0:m-1);
X=x./1000;

YM=0;
for i=3:n
    if (Y(i) > YM)
        YM = Y(i);
    end;
end;

X2=[0.02 0.02];
Y2=[0 YM];

plot(X,Y,X2,Y2)
axis([0.001 0.5 0 YM]);

%**************************************************************************
```

APPENDIX C
MULTI-CYCLE ENGINE MODEL SOURCE CODE

The Fortran 95 source code for the engine model is contained in this appendix, and can also be found on the CD accompanying this dissertation. This code is written using the extended precision data types available in the NAS compiler, but can be adapted for use in other compilers by changing the definition of the `(ep)` data type in the `rmap_globals` module. Syntax help for running the program can be viewed by running the program with the help command: `simulate.exe /?`

A brief description of the modules, subroutines, and functions found in each source file follows:

```
rmap.f90
    Program Simulate                Main program
        Uses:                       ISO_Varying_String
                                    RMap_Globals
                                    F2kCLI
                                    User_Set_Generator
rmap_globals.f90
    Module RMap_Globals             Global variable declarations
help.f90
    Subroutine RMap_Help            Displays CLI syntax help
input.f90
    Subroutine Input                Sets up parameters and reads input files
        Uses:                       RMap_Globals
initialize.f90
    Subroutine Initialize           Initializes variables for individual cycle
        Uses:                       RMap_Globals
model_loop.f90
    Subroutine Model                Calls cycle simulation subroutine and writes output to file
        Uses:                       RMap_Globals
                                    Random_Normal_Mod
        Arguments (in):             i, outunit1, outunit2
cycle.f90
    Subroutine Cycle_Simulation     Simulates an engine cycle
        Uses:                       RMap_Globals
        Arguments (in):             outunit
        Arguments (out):            HeatRelease, HeatRelease_alt
energy.f90
    Subroutine Energy               Applies the first law of thermodynamics
        Uses:                       RMap_Globals
        Arguments (in):             i
e_given_T_p_FOR.f90
    Function e_given_T_p_FOR        Looks up internal energy from T, P, Fuel/Oxygen ratio
        Uses:                       RMap_Globals
        Arguments (in):             T, p, phi, zone
T_given_e_p_FOR.f90
    Function T_given_e_p_FOR        Looks up T from internal energy, P, Fuel/Oxygen ratio
        Uses:                       RMap_Globals
        Arguments (in):             eint, p, phi, zone
T_given_e_p_resfrac
    Function T_given_e_p_resfrac    Iterates for T of initial mixture given P, residual fraction,
and temperatures of both residual gas and fresh charge
        Uses:                       RMap_Globals
        Arguments (in):             P
        Arguments (in/out):         T_guess1,T_guess2
burn.f90
    Subroutine Burn                 Uses turbulent mass entrainment model to determine mass
fraction burned
        Uses:                       RMap_Globals
        Arguments (in):             P, V, Phi
        Arguments (out):            SL, SL0, dmedt, dmbdt
wiebe.f90
```

```
    Subroutine Wiebe                Uses Wiebe function to determine mass fraction burned
        Uses:                       RMap_Globals
heat_transfer_rate.f90
    Function heat_transfer_rate     Determines wall heat transfer rate
        Uses:                       RMap_Globals
        Arguments (in):             i
update.f90
    Subroutine update               Updates values for iteration
        Uses:                       RMap_Globals
volume_function.f90
    Function volume_function        Calculates cylinder volume
        Uses:                       RMap_Globals
        Arguments (in):             angle
HR.f90
    Function Heat_Release           Calculates cycle heat release
        Uses:                       RMap_Globals
logsettings.f90
    Subroutine RMap_Log             Writes parameters to log file
        Uses:                       RMap_Globals
        Arguments (in):             FileName
BKFileIO.f90
    Subroutine BKInFile             Opens an existing file for input
        Arguments (in/out):         FileIn
        Arguments (out):            funit
    Subroutine BKOutFile            Creates a new file for output
        Arguments (in/out):         FileOut
        Arguments (out):            funit
    Subroutine BKFClose             Closes specified files
        Arguments (in):             n, funit
    Subroutine BKIOErr              Returns text for Fortran I/O error status (NAS Compiler)
        Uses:                       FORTRAN_IO_ERRORS
        Arguments (in):             ios
char_conv.f90
    Subroutine CharToInt            Converts string to integer
        Uses:                       Kinds
        Arguments (in):             InString
        Arguments (out):            OutNum
    Subroutine CharToReal           Converts string to real number
        Uses:                       Kinds
        Arguments (in):             InString
        Arguments (out):            OutNum
    Subroutine CharToDouble         Converts string to double precision number
        Uses:                       Kinds
        Arguments (in):             InString
        Arguments (out):            OutNum
    Subroutine CharToExtended       Converts string to extended precision number
        Uses:                       Kinds
        Arguments (in):             InString
        Arguments (out):            OutNum
    Subroutine CharToComplex        Converts string to complex number
        Uses:                       Kinds
        Arguments (in):             InString
        Arguments (out):            OutNum
    Subroutine CharToDblComplex     Converts string to double precision complex number
        Uses:                       Kinds
        Arguments (in):             InString
        Arguments (out):            OutNum
    Subroutine IntToChar            Converts integer to string
        Uses:                       Kinds
        Arguments (in):             InNum
        Arguments (out):            OutString
    Subroutine RealToChar           Converts real number to string
        Uses:                       Kinds
        Arguments (in):             InNum
        Arguments (out):            OutString
    Subroutine DoubleToChar         Converts double precision number to string
        Uses:                       Kinds
        Arguments (in):             InNum
        Arguments (out):            OutString
```

```
    Subroutine ExtendedToChar       Converts extended precision number to string
        Uses:                       Kinds
        Arguments (in):             InNum
        Arguments (out):            OutString
    Subroutine ComplexToChar        Converts complex number to string
        Uses:                       Kinds
        Arguments (in):             InNum
        Arguments (out):            OutString
    Subroutine DblComplexToChar     Converts double precision complex number to string
        Uses:                       Kinds
        Arguments (in):             InNum
        Arguments (out):            OutString
thermo_u.data                       Contains lookup table data for unburned gases
thermo_b.data                       Contains lookup table data for burned gases
```

A description of each global variable is found where it is declared in the `rmap_globals` module; local variables are commented where declared as well. External modules required are `f2kcli` (78) and `randlib90` (79) available from http://www.winteracter.com/f2kcli/ and http://biostatistics.mdanderson.org/SoftwareDownload/SingleSoftware.aspx?Software_Id=27 respectively. Both are available royalty free for non-commercial use.

```
*** BEGIN FILE RMAP.F90 ***

!  rmap.f90
!
!   Brian Kaul
!   June 2004
!   Fuels, Engines, and Emissions Research Center
!   Oak Ridge National Laboratory
!
!   Use cycle simulation model to simulate cyclic variability
!   Model algorithm adapted from F77 code by Kalyan

Program Simulate
    Use ISO_Varying_String
    Use RMap_Globals
    Use F2kCLI
    Use User_Set_Generator, Only: Time_Set_Seeds

!   Variables

    Implicit None

!   i,j,l = loop indices; ios = I/O Status indicator; funit = unit number for output file
!   nargs = number of command line arguments program is executed with
    Integer :: i,j,l,ios,nargs
!   nfiles = number of files to use
    Integer, Parameter :: nfiles = 4
!   ArgChar = Character variable for command line argument being considered
    Character(Len=20) :: ArgChar,istr
!   RunName = title given to this simulation
    Character(Len=50) :: RunName
!   fname = file names
    Character(Len=60), Dimension (nfiles) :: fname
    Integer, Dimension (nfiles) :: funit
!   LogSettings = if true, settings will be written to log file; Defaults = use default parameter
values if true
    Logical :: LogSettings,Defaults

    LogSettings = .False.
    Defaults = .True.
    CycleOutput = .False.
    WiebeFcn = .False.

!   Default Settings
    RunName='Simulation'
    n = 10000
    RPM = 1000.0
    equiv_ratio_in = 1.0
    equiv_ratio_noise = 0.010
    EGRIn = 0.0
    EGRNoise = 0.0
    ResIn = 0.14
    ResNoise = 0.0
    Spark_Advance = 10
    NType = 1
    EngineType = 'RIC'
    FuelType = 'I'
    u_prime_in = 0.05
    u_prime_noise = 0.0
    MAP = 101325.0
    T_wall = 450.0
    T_fresh = 300.0
    T_EGR = 400.0
    retard_factor = 1.0
    laminar_fraction = 0.05
    cr_factor = 1.0
    ht_rf = 1.0

!   Parse command line arguments and set parameters accordingly
```

```
    nargs = Command_Argument_Count()
    If (nargs > 0) Then
        i = 1
        Do
            Call Get_Command_Argument(number=i,value=ArgChar)
            If ((Index(ArgChar,'-',back=.false.) == 1).or.(Index(ArgChar,'/',back=.false.) == 1))
Then
                Select Case (Extract(ArgChar,2,2))
                    Case ('a','A')                          ! MAP
                        Defaults=.False.
                        i = i + 1
                        Call Get_Command_Argument(number=i,value=ArgChar)
                        Call CharToExtended(ArgChar,MAP)
                    Case ('c','C')                          ! Number of cycles/Compression Ratio
Factor
                        Defaults = .False.
                        i = i + 1
                        If ((Extract(ArgChar,3,3) == 'f') .or. (Extract(ArgChar,3,3) == 'F'))
Then
                            Call Get_Command_Argument(number=i,value=ArgChar)
                            Call CharToExtended(ArgChar,cr_factor)
                        Else
                            Call Get_Command_Argument(number=i,value=ArgChar)
                            Call CharToInt(ArgChar,n)
                        EndIf
                    Case ('e','E')                          ! EGR
                        Defaults = .False.
                        i = i + 1
                        If ((Extract(ArgChar,3,3) == 'n') .or. (Extract(ArgChar,3,3) == 'N'))
Then
                            Call Get_Command_Argument(number=i,value=ArgChar)
                            Call CharToExtended(ArgChar,EGRNoise)
                        ElseIf ((Extract(ArgChar,3,3) == 't') .or. (Extract(ArgChar,3,3) == 'T'))
Then
                            Call Get_Command_Argument(number=i,value=ArgChar)
                            Call CharToExtended(ArgChar,T_EGR)
                        Else
                            Call Get_Command_Argument(number=i,value=ArgChar)
                            Call CharToExtended(ArgChar,EGRIn)
                        EndIf
                    Case ('f','F')                          ! Fuel Type/Fresh Charge Temp
                        Defaults=.False.
                        i = i + 1
                        If ((Extract(ArgChar,3,3) == 't') .or. (Extract(ArgChar,3,3) == 'T'))
Then
                            Call Get_Command_Argument(number=i,value=ArgChar)
                            Call CharToExtended(ArgChar,T_Fresh)
                        Else
                            Call Get_Command_Argument(number=i,value=ArgChar)
                            FuelType = ArgChar
                        EndIf
                    Case ('h','H','?')
                        If ((Extract(ArgChar,3,3) == 'r') .or. (Extract(ArgChar,3,3) == 'R'))
Then
                            i = i + 1
                            Call Get_Command_Argument(number=i,value=ArgChar)
                            Call CharToExtended(ArgChar,ht_rf)
                        Else
                            Call RMap_Help                  ! Display help and exit
                            Stop
                        EndIf
                    Case ('l','L')                          ! Enable Settings Log
                        If ((Extract(ArgChar,3,3) == 'f') .or. (Extract(ArgChar,3,3) == 'F'))
Then
                            i = i + 1
                            Call Get_Command_Argument(number=i,value=ArgChar)
                            Call CharToExtended(ArgChar,laminar_fraction)
                        Else
                            LogSettings = .True.
```

```
                    EndIf
                Case ('m','M')                          ! Engine Type
                    Defaults=.False.
                    i = i + 1
                    Call Get_Command_Argument(number=i,value=ArgChar)
                    EngineType = ArgChar
                Case ('n','N')                          ! Name of simulation/Noise Type
                    Defaults = .False.
                    i = i + 1
                    If ((Extract(ArgChar,3,3) == 't') .or. (Extract(ArgChar,3,3) == 'T'))
Then
                        Call Get_Command_Argument(number=i,value=ArgChar)
                        Call CharToInt(ArgChar,NType)
                    Else
                        Call Get_Command_Argument(number=i,value=RunName)
                    EndIf
                Case ('o','O')                          ! Individual Cycle Output Files
                    CycleOutput = .True.
                Case ('p','P')                          ! Phi
                    Defaults = .False.
                    i = i + 1
                    If ((Extract(ArgChar,3,3) == 'n') .or. (Extract(ArgChar,3,3) == 'N'))
Then
                        Call Get_Command_Argument(number=i,value=ArgChar)
                        Call CharToExtended(ArgChar,equiv_ratio_noise)
                    Else
                        Call Get_Command_Argument(number=i,value=ArgChar)
                        Call CharToExtended(ArgChar,equiv_ratio_in)
                    EndIf
                Case ('r','R')                          ! Residual Fraction/Flame Speed
Retardation
                    Defaults = .False.
                    i = i + 1
                    If ((Extract(ArgChar,3,3) == 'n') .or. (Extract(ArgChar,3,3) == 'N'))
Then
                        Call Get_Command_Argument(number=i,value=ArgChar)
                        Call CharToExtended(ArgChar,ResNoise)
                    ElseIF ((Extract(ArgChar,3,3) == 'f') .or. (Extract(ArgChar,3,3) == 'F'))
Then
                        Call Get_Command_Argument(number=i,value=ArgChar)
                        Call CharToExtended(ArgChar,retard_factor)
                    Else
                        Call Get_Command_Argument(number=i,value=ArgChar)
                        Call CharToExtended(ArgChar,ResIn)
                    EndIf
                Case ('s','S')                          ! Engine RPM
                    Defaults=.False.
                    i = i + 1
                    Call Get_Command_Argument(number=i,value=ArgChar)
                    Call CharToReal(ArgChar,RPM)
                Case ('t','T')                          ! Ignition timing
                    Defaults = .False.
                    i = i + 1
                    Call Get_Command_Argument(number=i,value=ArgChar)
                    Call CharToReal(ArgChar,Spark_Advance)
                Case ('u','U')                          ! Turbulent Intensity
                    Defaults=.False.
                    i = i + 1
                    If ((Extract(ArgChar,3,3) == 'n') .or. (Extract(ArgChar,3,3) == 'N'))
Then
                        Call Get_Command_Argument(number=i,value=ArgChar)
                        Call CharToReal(ArgChar,u_prime_noise)
                    Else
                        Call Get_Command_Argument(number=i,value=ArgChar)
                        Call CharToReal(ArgChar,u_prime_in)
                    EndIf
                Case ('w','W')                          ! Wall Temp/Wiebe Fcn
                    Defaults=.False.
```

```
                        If ((Extract(ArgChar,3,3) == 'f') .or. (Extract(ArgChar,3,3) == 'F'))
Then
                            WiebeFcn = .True.
                        Else
                            i = i + 1
                            Call Get_Command_Argument(number=i,value=ArgChar)
                            Call CharToExtended(ArgChar,T_wall)
                        EndIf
                    Case Default
                        Write (unit=*,fmt=105) 'Invalid Argument: ',ArgChar
                        If (Defaults .and. i == nargs) Then
                            Call RMap_Help
                            Stop
                        Else
                            Do
                                Write (unit=*,fmt=100) 'Continue? (Y/N)'
                                Read (unit=*,fmt='(a)') ArgChar
                                Select Case (ArgChar)
                                    Case ('y','Y')
                                        Exit
                                    Case ('n','N')
                                        Stop
                                    Case Default
                                        Write (unit=*,fmt=100) 'Invalid entry.'
                                EndSelect
                            EndDo
                        EndIf
                EndSelect
            Else
                Write (unit=*,fmt=105) 'Invalid Argument: ',ArgChar
                If (Defaults .and. i == nargs) Then
                    Call RMap_Help
                    Stop
                Else
                    Do
                        Write (unit=*,fmt=100) 'Continue? (Y/N)'
                        Read (unit=*,fmt='(a)') ArgChar
                        Select Case (ArgChar)
                            Case ('y','Y')
                                Exit
                            Case ('n','N')
                                Stop
                            Case Default
                                Write (unit=*,fmt=100) 'Invalid entry.'
                        EndSelect
                    EndDo
                EndIf
            EndIf
            i = i + 1
            If (i > nargs) Exit
        EndDo
    EndIf

    soc = 0.0 - Spark_Advance*pi/180.0_ep

!   Output file = RunName.csv, Log file = RunName.set
    fname(1)=Trim(RunName) // '.csv'
    fname(2)=Trim(RunName) // '.set'
    fname(4)=Trim(RunName) // '-motored.csv'

    Call Time_Set_Seeds                                     !   Reset Random Number Generator
Seeds
    Call Input                                                      !   Read in thermo tables,
etc
    Call BKOutFile(fname(1),funit(1))           !   Open output file

    Allocate(HR(n),HR2(n))

    Write (unit=funit(1),fmt=110)
```

```
!   Find the initial motored pressure trace for Woschni
    equiv_ratio_fresh = equiv_ratio_In
    EGR_fraction = EGRIn
    residual_fraction = ResIn
    TurbLevel = u_prime_in
    FO_ratio_fresh = stoic_coeff_O2*mwO2*equiv_ratio_fresh/(0.21*mwF*AFS)
    Motored = .True.

    Call Initialize
    Call BKOutFile(fname(4),funit(4))
    i = 1

    Call Cycle_Simulation(HR(1),HR2(1),funit(4))

    pm = pressure
    Call BKFClose(1,funit(4))
    Motored = .False.

!   Loop the model for the given number of cycles
    Do i=1,n
        If (CycleOutput) Then
            Call IntToChar(i,istr)
            fname(3)=Trim(RunName) // '-' // Trim((AdjustL(istr))) // '.csv'
            Call BKOutFile(fname(3),funit(3))
            Write (unit=funit(3),fmt=130)
        EndIf
    !   pressure=>press(:,i)                        !   Point pressure to the current cycle's
column in press
        pressure_sum = pressure_sum + pressure
        Call Model(i,funit(1),funit(3))
        If (CycleOutput) Call BKFClose(1,funit(3))
    EndDo

    pressure_sum = pressure_sum/n

    Call BKFClose(1,funit(1))
    If (LogSettings == .True.) Call RMap_Log(fname(2))

    fname(3)=Trim(RunName) // '-avg.csv'
    Call BKOutFile(fname(3),funit(3))
    Write (unit=funit(3),fmt=140)
    Do i=1,nsteps
        theta = 0.1*i
        Write (unit=funit(3),fmt=150) theta,pressure_sum(i)
    EndDo
    Call BKFClose(1,funit(3))

    Write (unit=*,fmt=100) 'Finished'

!   Formatting
    100 Format (1x,a)
    105 Format (1x,2a)
    110 Format (1x,'Cycle,Heat Release (P-V),Heat Release (de/dt),Gross Fuel Heat
Release,Combustion Efficiency,Air In,Fuel In,Phi In,Total Moles')
    120 Format (1x,i5,',',E15.8E3,',',E15.8E3)
    130 Format
(1x,'Theta,T_u,T_b,P,burn_fraction,V_b/volume,volume,SL,SL0,rho_u,Area_f,u_prime,flame_radius,dmbd
t,h,r_cyl,dvdt,ht_rate,energy_internal_u,energy_internal_b')
    140 Format (1x,'Crank Angle (Deg),Pressure')
    150 Format (1x,F7.1,',',E15.8E3)

End Program Simulate

*** END FILE RMAP.F90 ***
```

```
*** BEGIN FILE RMAP_GLOBALS.F90 ***

!   rmap_globals.f90
Module RMap_Globals
!   If not using NAS Fortran compiler, remove the Kinds USE and use commented Integer declaration
below
!   (in this and other files that USE KINDS)
    Use Kinds, Only extended,double,single               !   IEEE 754-1985 standard KINDS
nomenclature
    Implicit None

!   Global Variables

!   Unless otherwise noted, all lengths in m, volumes in m^3, time in s, velocities in m/s, mass
in kg

    Integer, Parameter :: ep=extended,dp=double,sp=single !  Switch to line below if not USE-ing
KINDS
!   Integer, Parameter ::
sp=Selected_Real_Kind(5,20),dp=Selected_Real_Kind(10,40),ep=Selected_Real_Kind(18,100)
    Real(ep), Parameter :: pi=3.14159265358979323846264 3

!   equiv_ratio_in = Mean input Equivalence Ratio; equiv_ratio_noise = Standard Deviation of noise
on Equivalence Ratio
!   EGRIn = Mean EGR Fraction; EGRNoise = Standard Deviation of noise on EGR Fraction
!   ResIn = Mean Residual Fraction; ResNoise = Standard Deviation of noise on Residual Fraction
    Real(ep) :: equiv_ratio_in,equiv_ratio_noise,EGRIn,EGRNoise,ResIn,ResNoise,equiv_ratio_u
!   mass_fuel = total fuel mass in cylinder; mass_air = total air mass in cylinder; nT = total
moles in cylinder
    Real(ep) :: mass_fuel,mass_air,nT,retard_factor,laminar_fraction,cr_factor
!   NType = type of noise to use (0 = none, 1 = Gaussian); n = number of cycles
    Integer :: NType,n
!   HCR = fuel H/C Ratio; mwF = molecular weight of fuel; mwA = molecular weight of air; mwEGR =
molecular weight of EGR
!   mwCO2 = molecular weight of CO2; mwH2O = molecular weight of water; mwO2 = molecular weight of
O2
    Real(sp), Parameter :: mwEGR=30.41,mwCO2=44.01,mwH2O=18.02,mwN2=28.01,mwO2=32.00,mwA=28.85
    Real(sp) :: AFS,HCR,mwF,soc,evo,LHV
    Character (Len=1) :: FuelType
!   mwflue = molecular weight of flue gases; mwC = molecular weight of Carbon; mwH = molecular
weight of Hydrogen
    Real(sp), Parameter :: mwflue=30.25,mwC=12.01,mwH=1.01
!   eps, epsi, epsl = "epsilon" values used for convergence tolerances, etc.
!   AFS = Stoichiometric air/fuel ratio
    Real(sp), Parameter :: eps=1.0e-3,epsi=1.0e-4,epsl=1.0e-5
!   x,y = (CxHy) number of effective carbon/hydrogen atoms in fuel molecule, for stoichiometry
    Real(sp) :: x,y
!   HR = Heat Release
    Real(ep), Dimension(:), Allocatable :: HR,HR2
!   spark_advance = spark timing (degrees BTDC)
    Real(sp) :: spark_advance
!   enginetype = What engine is being simulated? (CFR/RIC for CFR/Ricardo)
    Character (Len=3) :: EngineType

!   mwEGR = 0.15*mwCO2+0.85*mwN2
!   mwflue = 0.47*mwCO2+0.53*mwH2O

!   nsteps = number of steps to go through per cycle (function of theta & dtheta)
    Integer :: nsteps

!   Cylinder Geometry - all lengths in meters, volumes in m^3
!   volume_max = volume at BDC; clearance_volume = volume at TDC; displacement = displacement
volume
!   Compression_ratio, stroke, bore = compression ratio, stroke, & bore of engine (intuitive)
    Real(sp) :: volume_max,clearance_volume,displacement,compression_ratio,stroke,bore
!   crank_radius = 2*stroke; rod_length = length of connecting rod
!   rod_crank_ratio = rod_length/crank_radius; stroke_bore_ratio = stroke/bore
    Real(sp) :: crank_radius,rod_length,rod_crank_ratio,stroke_bore_ratio
!   theta = crank angle (radians); dtheta = change in crank angle per step
```

```
!   h = height of cylinder; h_old = height at previous step; dh = change in height per step
!   dt = change in time per step (s)
    Real(ep) :: theta,dtheta,h,h_old,dh,dt

!   Operational Parameters
!   RPM = crankshaft revolutions per minute; theta_0 = crank angle at start of combustion for
Weibe fcn
!   theta_b = crank angle duration of combustion for Weibe fcn; piston_speed = speed (m/s) of
piston
!   mean_piston_speed = mean piston speed for a given RPM
    Real(sp) :: RPM,theta_0,theta_b,piston_speed,mean_piston_speed

!   Dependent variables : old & new states for unburned & burned zones

!   Geometric Variables
!   volume = volume of cylinder
!   area_heat_transfer = wall heat transfer surface area
    Real(ep) :: area_heat_transfer

!   Volumes
!   V_u = unburned volume; V_u_old = unburned volume at previous step
!   V_b = burned volume; V_b_old = burned volume at previous step
    Real(ep) :: V_u,V_u_old,V_b,V_b_old
    Real(ep), Dimension(:), Allocatable :: volume,dvdt

!   Charges
!   cmass = charge mass; cmass_u = unburned charge mass; cmass_u_old = unburned charge mass at
previous step
!   cmass_b = burned charge mass; cmass_b_old = burned charge mass at previous step
!   FO_ratio = (stoichiometric coefficient of O2)*(moles of fuel)/(moles of O2)
!   FO_ratio_b = FO_ratio for burned zone; FO_ratio_u = FO_ratio for unburned zone
!   equiv_ratio = equivalence ratio: AFS*(fuel mass/air mass)
    Real(ep) ::
cmass,cmass_u,cmass_u_old,cmass_b,cmass_b_old,FO_ratio,FO_ratio_b,FO_ratio_u,equiv_ratio

!   Temperatures
!   T_u = temperature in unburned zone; T_u_old = temperature in unburned zone at previous step
!   T_b = temperature in burned zone; T_b_old = temperature in burned zone at previous step
!   T_avg = average temperature of charge; T_wall = wall temperature
    Real(ep) :: T_u,T_u_old,T_b,T_b_old,T_avg,T_wall

!   Energies
!   energy_internal_u = internal energy of unburned zone; energy_internal_u_old =
energy_internal_u at previous step
!   energy_internal_b = internal energy of burned zone; energy_internal_b_old = energy_internal_b
at previous step
    Real(ep) ::
energy_internal_u,energy_internal_u_old,energy_internal_b,energy_internal_b_old,dedt

!   Pressures
!   pressure = points to pressure array for current cycle only, all steps
!   Real(ep), Dimension(:), Pointer :: pressure
!   pressure_old = pressure from previous step
    Real(ep), Dimension(:), Allocatable :: pressure_old
!   press = pressure for all cycles, steps
    Real(ep), Dimension(:), Allocatable :: pressure, pressure_sum, pm
    Real(ep) :: MAP

!   Standard Conditions
!   T_s, P_s = temperature, pressure at standard conditions (K, Pa)
!   nu_s = viscosity of air at standard conditions
    Real(ep), Parameter :: T_s=298.15, P_s=101325, nu_s=1.56e-5

!   Gas Constants
!   R_u, R_b = gas constant for unburned & burned zones
    Real(sp) :: R_u,R_b
!   gamma_u, gamma_b, gamma = Cp/Cv for unburned, burned zones, & mass-averaged
    Real(sp) :: gamma_u,gamma_b,gamma
!   R_univ = universal gas constant
```

```
    Real(sp), Parameter :: R_univ=8.314

!   Densities
!   rho_u, rho_b = densities for unburned & burned zones; rho_0 = unburned density at start of
combustion
    Real(ep) :: rho_u,rho_b,rho_0,rho_s,T_0,P_0

!   Tabulated thermodynamic properties

!   Table Range (b)
!   p_min_b, p_max_b = range of pressure values allowed for burned zone thermodynamic properties
lookup table
!   FO_ratio_min_b, FO_ratio_max_b = range of FO_ratio values allowed
!   Tmp_min_b, Tmp_max_b = range of temperature values allowed
    Real(sp) :: p_min_b,p_max_b,FO_ratio_min_b,FO_ratio_max_b,Tmp_min_b,Tmp_max_b

!   NTable (b)
!   ntable_p_b, ntable_FO_ratio_b, ntable_T_b = size of burned table for p, FO_ratio, T
    Integer :: ntable_p_b,ntable_FO_ratio_b,ntable_T_b

!   Table Thermo (b)
!   table_p_b, table_FO_ratio_b, table_T_b = P, FO_ratio, T values for which properties are
tabulated
    Real(sp), Dimension(:), Allocatable :: table_p_b,table_FO_ratio_b,table_T_b
!   thermo_b = burned thermodynamic properties table
    Real(sp), Dimension(:,:,:), Allocatable :: thermo_b

!   Table Range (u)
!   p_min_u, p_max_u = range of pressure values allowed for unburned zone thermodynamic properties
lookup table
!   FO_ratio_min_u, FO_ratio_max_u = range of FO_ratio values allowed
!   Tmp_min_u, Tmp_max_u = range of temperature values allowed
    Real(sp) :: p_min_u,p_max_u,FO_ratio_min_u,FO_ratio_max_u,Tmp_min_u,Tmp_max_u

!   NTable (u)
!   ntable_p_b, ntable_FO_ratio_b, ntable_T_b = size of unburned table for p, FO_ratio, T
    Integer :: ntable_p_u,ntable_FO_ratio_u,ntable_T_u

!   Table Thermo (u)
!   table_p_b, table_FO_ratio_b, table_T_b = P, FO_ratio, T values for which properties are
tabulated
    Real(sp), Dimension(:), Allocatable :: table_p_u,table_FO_ratio_u,table_T_u
!   thermo_u = unburned thermodynamic properties table
    Real(sp), Dimension(:,:,:), Allocatable :: thermo_u

!   RNumerical
!   relax_fac = relaxation factor for iterations of P,V,burn_fraction
    Real(sp) :: relax_fac

!   Heat Transfer
!   ht_rate = heat transfer rate; ht_rate_u = heat transfer rate from unburned zone
!   ht_rate_b = heat transfer rate from burned zone; ht_rate_u_old, ht_rate_b_old = from previous
step
!   ht_rf = heat transfer reduction factor (multiplicative on ht_rate)
    Real(ep) :: ht_rate,ht_rate_u,ht_rate_u_old,ht_rate_b,ht_rate_b_old,ht_rf

!   Burning Model
!   flame_radius = effective radius of spherical flame to give current burned volume
!   mass_ent = mass entrained in flame; flame_radius_old, mass_ent_old = from previous step
!   u_prime_0 = turbulence intensity at start of combustion; li_0 = length scale at start of
combustion
!   area_f = flame front surface area (assumed smooth, spherical)
    Real(ep) :: flame_radius,flame_radius_old,mass_ent,mass_ent_old,u_prime,u_prime_0,area_f,li_0
    Real(sp) :: u_prime_in,u_prime_noise,TurbLevel
    Integer :: frscale
!   burn_fraction = burned mass fraction, burn_fraction_old = burned mass fraction from previous
iteration
!   burn_fraction_2 = burned mass fraction from iteration before previous iteration
    Real(ep) :: burn_fraction,burn_fraction_old,burn_fraction_2
```

```
!   Ignition = True after spark; Quenched = True if quenched; CycleOutput = True if output files
per cycle is enabled
    Logical :: Ignition,Quenched,CycleOutput,WiebeFcn,Motored
!   ahr_Ea = Ea/R_u; ahr_A = A; ahr_rate = A*exp(-
Ea/(R_u*T_avg))*molar_conc_fuel^m*molar_conc_o2^n (concentrations in mols/cm^3)
!   ahr_m = m; ahr_n = n; ahr_const = value of ahr_rate to compare
    Real(sp), Parameter :: ahr_Ea = 15098, ahr_A = 5.0e5, ahr_m = 0.25, ahr_n = 1.5, ahr_const =
1e-20
    Real(ep) :: ahr_rate

!   Residual Characteristics
!   residual_fraction = residual fraction; T_res = residual temperature
!   FO_ratio_res_u = FO_ratio of the portion of residual that was in unburned zone in previous
cycle
!   FO_ratio_res_b = FO_ratio of the portion of residual that was in burned zone in previous cycle
!   residual_fraction_b, residual_fraction_u = fraction of residual from burned/unburned zone in
previous cycle
    Real(ep) ::
residual_fraction,T_res,FO_ratio_res_b,residual_fraction_b,FO_ratio_res_u,residual_fraction_u

!   EGR Characteristics
!   EGR_fraction = EGR Fraction; T_EGR = EGR temperature
!   FO_ratio_EGR_u = FO_ratio of unburned portion of EGR
!   FO_ratio_EGR_b = FO_ratio of burned portion of EGR
!   EGR_fraction_b, EGR_fraction_u = fraction of EGR that is burned/unburned
    Real(ep) :: EGR_fraction,T_EGR,FO_ratio_EGR_b,EGR_fraction_b,FO_ratio_EGR_u,EGR_fraction_u

!   Fresh Charge
!   T_fresh = fresh charge temperature; FO_ratio_fresh = FO_ratio of fresh charge
!   equiv_ratio_fresh = equivalence ratio of fresh charge
    Real(ep) :: T_fresh,FO_ratio_fresh,equiv_ratio_fresh

!   Stoichiometry
!   fuel_wt_per_O2_moles = mass of fuel per moles of O2; stoic_coeff_O2 = stoichiometric
coefficient of O2
    Real(ep) :: fuel_wt_per_O2_moles,stoic_coeff_O2
!   O2_moles_fresh = moles of O2 in fresh charge; N2_moles_fresh = moles of N2 in fresh charge
!   fuel_moles_fresh = moles of fuel in fresh charge; O2_moles_res_u = moles of O2 in unburned
portion of residual
!   N2_moles_res_u = moles of N2 in unburned portion of residual
    Real(ep) :: o2_moles_fresh,n2_moles_fresh,fuel_moles_fresh,o2_moles_res_u,n2_moles_res_u
!   fuel_moles_res = moles of fuel in residual; O2_moles_res_b = moles of O2 in burned portion of
residual
!   N2_moles_res_b = moles of N2 in burned portion of residual; flue_moles_res_b = moles of flue
gas in burned portion of residual
!   stoic_coeff_flue = stoichiometric coefficient of flue gas
    Real(ep) :: fuel_moles_res,o2_moles_res_b,n2_moles_res_b,flue_moles_res_b,stoic_coeff_flue
!   O2_moles_EGR_u = moles of O2 in unburned portion of EGR; N2_moles_EGR_u = moles of N2 in
unburned portion of EGR
!   fuel_moles_EGR = moles of fuel in EGR; O2_moles_EGR_b = moles of O2 in burned portion of EGR
!   N2_moles_EGR_b = moles of N2 in burned portion of EGR
    Real(ep) :: o2_moles_egr_u,n2_moles_egr_u,fuel_moles_egr,o2_moles_egr_b,n2_moles_egr_b
!   flue_moles_EGR_b = moles of flue gas in burned portion of EGR; O2_moles = total moles of O2
!   N2_moles = total moles of N2; fuel_moles = total moles of fuel; flue_moles = total moles of
flue gas
!   fuel_conc = concentration of fuel; O2_conc = concentration of O2; flue_conc = concentration of
flue gas
    Real(ep) ::
flue_moles_egr_b,o2_moles,n2_moles,fuel_moles,flue_moles,fuel_conc,o2_conc,flue_conc
    Real(ep) :: fuel_V_conc, O2_V_conc
!   energy_internal_fresh = internal energy of fresh charge
!   energy_internal_res_b, energy_internal_res_u = internal energy of burned/unburned portion of
residual
    Real(ep) :: energy_internal_fresh,energy_internal_res_b,energy_internal_res_u
!   energy_internal_res = internal energy of total residual; energy_internal_EGR = internal energy
of total EGR
!   energy_internal_EGR_b, energy_internal_EGR_u = internal energy of burned/unburned portion of
EGR
```

```
    Real(ep) ::
energy_internal_res,energy_internal_egr_b,energy_internal_egr_u,energy_internal_egr

End Module RMap_Globals

*** END FILE RMAP_GLOBALS.F90 ***
```

```
*** BEGIN FILE HELP.F90 ***

Subroutine RMap_Help()
    Implicit None

    Write (unit=*,fmt=100) 'Simulates cyclic variability using Two-zone Cycle Simulation model'
    Write (unit=*,fmt=100)
    Write (unit=*,fmt=100) ' Syntax:'
    Write (unit=*,fmt=100) ' Simulate [ -h | -n name | -c cycles | -p equiv_ratio | -e
EGR_fraction |'
    Write (unit=*,fmt=100) '           -r residual_fraction | -pn equiv_ratio_noise | -en
EGR_noise |'
    Write (unit=*,fmt=100) '           -rn residual_noise | -nt noise_type | -t timing | -a MAP |'
    Write (unit=*,fmt=100) '           -u u_prime | -un u_prime_noise | -w T_wall | -s speed | -wf
|'
    Write (unit=*,fmt=100) '           -et T_EGR | -ft T_fresh | -rf retard_factor | -l | -o]'
    Write (unit=*,fmt=100)
    Write (unit=*,fmt=100) ' -h    Displays this syntax help message (also /?)'
    Write (unit=*,fmt=100) ' -c    Specify number of cycles to simulate (Default 10000)'
    Write (unit=*,fmt=100) ' -p    Specify mean input equivalence ratio (Default 1.0)'
    Write (unit=*,fmt=100) ' -pn   Specify standard deviation of noise on equivalence ratio
(Defualt 0.010)'
    Write (unit=*,fmt=100) ' -e    Specify mean EGR fraction (Default 0.0)'
    Write (unit=*,fmt=100) ' -en   Specify standard deviation of noise on EGR fraction (Default
0.000)'
    Write (unit=*,fmt=100) ' -r    Specify mean residual fraction (Default 0.14)'
    Write (unit=*,fmt=100) ' -rn   Specify standard deviation of noise on residual fraction
(Default 0.000)'
    Write (unit=*,fmt=100) ' -rf   Specify flame speed retardation multiplicative factor (Default
1.0)'
    Write (unit=*,fmt=100) ' -hr   Specify heat transfer reduction multiplicative factor (Default
1.0)'
    Write (unit=*,fmt=100) ' -nt   Specify type of noise (Default 1):'
    Write (unit=*,fmt=100) '           0 = None'
    Write (unit=*,fmt=100) '           1 = Gaussian'
    Write (unit=*,fmt=100) ' -m    Specify engine geometry to be used (Default RIC):'
    Write (unit=*,fmt=100) '           CFR = CFR Engine Geometry'
    Write (unit=*,fmt=100) '           RIC = Ricardo Engine Geometry'
    Write (unit=*,fmt=100) ' -f    Specify fuel to be used (Default I):'
    Write (unit=*,fmt=100) '           I = Iso-Octane'
    Write (unit=*,fmt=100) '           G = Gasoline'
    Write (unit=*,fmt=100) ' -t    Specify spark timing (degrees BTDC) (Default 10.0)'
    Write (unit=*,fmt=100) ' -a    Specify MAP pressure (Pa) (Default 101325.0)'
    Write (unit=*,fmt=100) ' -u    Specify turbulence level mean (fraction of piston speed)
(Default 0.1)'
    Write (unit=*,fmt=100) ' -un   Specify standard deviation of noise on turbulence level
(Default 0.000)'
    Write (unit=*,fmt=100) ' -w    Specify cylinder wall temperature (K) (Default 450.0)'
    Write (unit=*,fmt=100) ' -s    Specify engine speed (RPM) (Default 1000.0)'
    Write (unit=*,fmt=100) ' -et   Specify EGR temperature (K) (Default 400.0)'
    Write (unit=*,fmt=100) ' -ft   Specify fresh charge temperature (K) (Default 300.0)'
    Write (unit=*,fmt=100) ' -wf   Use Wiebe Function instead of combustion model'
    Write (unit=*,fmt=100) ' -l    Enables logging of settings to log file'
    Write (unit=*,fmt=100) ' -o    Enables output of individual cycle data'
    Write (unit=*,fmt=100)
    Write (unit=*,fmt=100) ' All arguments are optional; defaults will be used if ommitted.'
    Return
    100 Format (a)
End Subroutine RMap_Help

*** END FILE HELP.F90 ***
```

```
*** BEGIN FILE INPUT.F90 ***

Subroutine Input
    Use RMap_Globals

    Implicit None

!   Functions
    Real(ep) :: Volume_Function
!   Loop indices for input
    Integer :: i,j,k
    Integer, Dimension(2) :: funit

!   Geometric parameters (dimensions in meters)
    Select Case (EngineType)
        Case ('CFR','cfr','Cfr')
            bore = 0.0825
            stroke = 0.1143
            rod_length = 0.254
            compression_ratio = 9.0
            evo = 130*pi/180.0_ep
        Case ('RIC','ric','Ric')
            bore = 0.0848
            stroke =  0.088
            rod_length = 0.13619
            compression_ratio = 9.3
            evo = 130*pi/180.0_ep
        Case Default
            Print *,'Invalid engine geometry specified.  Using CFR geometry.'
            bore = 0.0825
            stroke = 0.1143
            rod_length = 0.254
            compression_ratio = 9.0
            evo = 130*pi/180.0_ep
    EndSelect

    Select Case (FuelType)
        Case ('I','i') ! Iso-octane
            HCR=2.25
            mwF=114.23
            AFS=15.13
            LHV=44.3
        Case ('G','g','R','r') ! Gasoline/Indolene
            HCR=1.87
            mwF=114
            AFS=14.6
            LHV=44.0
        Case ('P','p') ! Propane
            HCR=8/3.0
            mwF=44.096
            AFS=15.544
            LHV=46.4
        Case ('M','m') ! Methanol
            HCR=4.0 ! Valid for oxygenated fuel?
            mwF=32.040
            AFS=6.418
            LHV=20.0
        Case ('E','e') ! Ethanol
            HCR=3.0 ! Valid for oxygenated fuel?
            mwF=46.07
            AFS=8.927
            LHV=26.9
        Case Default
            Print *,'Unknown fuel specified.  Using Isooctane parameters.'
            HCR=2.25
            mwF=114.23
            AFS=15.13
            LHV=44.3
    EndSelect
```

```
    stroke_bore_ratio = stroke/bore
    displacement = 0.25*pi*stroke*bore**2
    clearance_volume = displacement/(cr_factor*compression_ratio-1)
    volume_max = cr_factor*compression_ratio*clearance_volume
    crank_radius = 0.5*stroke
    rod_crank_ratio = rod_length/crank_radius

!   Operational parameters

    theta_0 = -20*(Atan(1.0_ep)/45.0_ep)
    theta_b = 60*(Atan(1.0_ep)/45.0_ep)

    mean_piston_speed = 2*stroke*RPM*pi/30.0_ep

!   Thermodynamic properties

!   Stoichiometry
    x = mwF/(mwC+HCR*mwH)
    y = HCR*x

    stoic_coeff_O2 = x+0.25*y
    stoic_coeff_flue = x+0.5*y

    Call BKInFile('thermo_b.data',funit(1))
    Call BKInFile('thermo_u.data',funit(2))

    Read(unit=funit(1),fmt=*) ntable_p_b,ntable_FO_ratio_b,ntable_T_b
    Read(unit=funit(2),fmt=*) ntable_p_u,ntable_FO_ratio_u,ntable_T_u

    Allocate(table_p_b(ntable_p_b),table_p_u(ntable_p_u))
    Allocate(table_FO_ratio_b(ntable_FO_ratio_b),table_FO_ratio_u(ntable_FO_ratio_u))
    Allocate(table_T_b(ntable_T_b),table_T_u(ntable_T_u))
    Allocate(thermo_b(ntable_p_b,ntable_FO_ratio_b,ntable_T_b),thermo_u(ntable_p_u,ntable_FO_ratio
_u,ntable_T_u))

    Read(unit=funit(1),fmt=*) p_min_b,p_max_b
    Read(unit=funit(1),fmt=*) FO_ratio_min_b,FO_ratio_max_b
    Read(unit=funit(1),fmt=*) Tmp_min_b,Tmp_max_b
    Read(unit=funit(1),fmt=*)
(((thermo_b(i,j,k),i=1,ntable_p_b),j=1,ntable_FO_ratio_b),k=1,ntable_T_b)

    Read(unit=funit(2),fmt=*) p_min_u,p_max_u
    Read(unit=funit(2),fmt=*) FO_ratio_min_u,FO_ratio_max_u
    Read(unit=funit(2),fmt=*) Tmp_min_u,Tmp_max_u
    Read(unit=funit(2),fmt=*)
(((thermo_u(i,j,k),i=1,ntable_p_u),j=1,ntable_FO_ratio_u),k=1,ntable_T_u)

    Call BKFClose(2,funit)

    theta = -pi
    dtheta = 0.1*(pi/180.0_ep)
    nsteps = Int(2*pi/dtheta)+1
    dt = dtheta*60/(RPM*2*pi)

    Allocate(pressure(nsteps),pressure_old(nsteps),pressure_sum(nsteps),pm(nsteps))
    Allocate(volume(nsteps),dvdt(nsteps))

    volume(1) = Volume_Function(theta)
    Do i=2,nsteps
        theta = theta + dtheta
        volume(i) = Volume_Function(theta)
    EndDo
    dvdt(1) = (volume(2) - volume(nsteps))/(2*dtheta)
    Do i=2,nsteps-1
        dvdt(i) = (volume(i+1) - volume(i-1))/(2*dtheta)
    EndDo
    dvdt(nsteps) = (volume(1) - volume(nsteps-1))/(2*dtheta)
    theta = -pi
```

```
    pressure(1) = 101325.

    Return
End Subroutine Input

*** END FILE INPUT.F90 ***
```

```
*** BEGIN FILE INITIALIZE.F90 ***

Subroutine Initialize
    Use RMap_Globals

    Implicit None

!   Functions
    Real(ep) :: e_given_T_p_FOR,Heat_Transfer_Rate,T_given_e_p_resfrac
!   Intermediate values
    Real(ep) :: FO_ratio_num,FO_ratio_den,o2_wt_per_o2_moles,n2_wt_per_o2_moles,wt_per_o2_moles
    Real(ep) :: t_guess1,t_guess2

!   Residual characteristics

    If (T_res /= 0) Then     !   If properties from previous cycle, use them
        T_res = burn_fraction*T_b + (1-burn_fraction)*T_u
        residual_fraction_b = burn_fraction
        FO_ratio_res_b = FO_ratio_b
        residual_fraction_u = 1 - residual_fraction_b
        FO_ratio_res_u = FO_ratio_u
    Else                         !   No previous cycle - initial guesses for first cycle
        T_res = 775.0
        residual_fraction_b = 0.99
        FO_ratio_res_b = 0.9
        residual_fraction_u = 1 - residual_fraction_b
        FO_ratio_res_u = 0.9
    EndIf

    burn_fraction = 0.0
    pressure = 0.0
    pressure(1) = MAP

!   EGR characteristics

    EGR_fraction_b = 0.9
    FO_ratio_EGR_b = 0.9
    EGR_fraction_u = 1 - EGR_fraction_b
    FO_ratio_EGR_u = 0.88

!   Mixture composition

    FO_ratio_num = 0.0
    FO_ratio_den = 0.0

!   fresh mixture

    fuel_wt_per_O2_moles = FO_ratio_fresh*mwF/stoic_coeff_O2
    O2_wt_per_O2_moles = mwO2
    N2_wt_per_O2_moles = mwN2*3.76

    wt_per_O2_moles = fuel_wt_per_O2_moles+O2_wt_per_O2_moles+N2_wt_per_O2_moles
    O2_moles_fresh = (1-residual_fraction-EGR_fraction)/wt_per_O2_moles

    FO_ratio_num = FO_ratio_num + FO_ratio_fresh*O2_moles_fresh
    FO_ratio_den = FO_ratio_den + O2_moles_fresh

    N2_moles_fresh = O2_moles_fresh*3.76
    fuel_moles_fresh = O2_moles_fresh*FO_ratio_fresh/stoic_coeff_O2

!   unburned residual fraction

    fuel_wt_per_O2_moles = FO_ratio_res_u*mwF/stoic_coeff_O2
    wt_per_O2_moles = fuel_wt_per_O2_moles+ O2_wt_per_O2_moles+N2_wt_per_O2_moles
    O2_moles_res_u  = residual_fraction*residual_fraction_u/wt_per_O2_moles

    FO_ratio_num = FO_ratio_num + FO_ratio_res_u*O2_moles_res_u
    FO_ratio_den = FO_ratio_den + O2_moles_res_u
```

```
    N2_moles_res_u = O2_moles_res_u*3.76
    fuel_moles_res = O2_moles_res_u*FO_ratio_res_u/stoic_coeff_O2

!   burned residual fraction

    fuel_wt_per_O2_moles = FO_ratio_res_b*mwF/stoic_coeff_O2
    wt_per_O2_moles = fuel_wt_per_O2_moles+O2_wt_per_O2_moles+N2_wt_per_O2_moles
    O2_moles_res_b = residual_fraction*residual_fraction_b/wt_per_O2_moles

    FO_ratio_num = FO_ratio_num + FO_ratio_res_b*O2_moles_res_b
    FO_ratio_den = FO_ratio_den + O2_moles_res_b

    N2_moles_res_b = O2_moles_res_b*3.76
    flue_moles_res_b= O2_moles_res_b*FO_ratio_res_b*stoic_coeff_flue/stoic_coeff_O2
    O2_moles_res_b = O2_moles_res_b*(1-FO_ratio_res_b)

!   unburned EGR fraction

    fuel_wt_per_O2_moles = FO_ratio_EGR_u*mwF/stoic_coeff_O2
    wt_per_O2_moles = fuel_wt_per_O2_moles+O2_wt_per_O2_moles+N2_wt_per_O2_moles
    O2_moles_EGR_u = EGR_fraction*EGR_fraction_u/wt_per_O2_moles

    FO_ratio_num = FO_ratio_num + FO_ratio_EGR_u*O2_moles_EGR_u
    FO_ratio_den = FO_ratio_den + O2_moles_EGR_u

    N2_moles_EGR_u = O2_moles_EGR_u*3.76
    fuel_moles_EGR = O2_moles_EGR_u*FO_ratio_EGR_u/stoic_coeff_O2

!   burned EGR fraction

    fuel_wt_per_O2_moles = FO_ratio_EGR_b*mwF/stoic_coeff_O2
    wt_per_O2_moles = fuel_wt_per_O2_moles+O2_wt_per_O2_moles+N2_wt_per_O2_moles
    O2_moles_EGR_b = EGR_fraction*EGR_fraction_b/wt_per_O2_moles

    FO_ratio_num = FO_ratio_num + FO_ratio_EGR_b*O2_moles_EGR_b
    FO_ratio_den = FO_ratio_den + O2_moles_EGR_b

    N2_moles_EGR_b = O2_moles_EGR_b*3.76
    flue_moles_EGR_b = O2_moles_EGR_b*FO_ratio_EGR_b*stoic_coeff_flue/stoic_coeff_O2
    O2_moles_EGR_b = O2_moles_EGR_b*(1_ep-FO_ratio_EGR_b)

!   Overall moles

    O2_moles = O2_moles_fresh+O2_moles_res_u+O2_moles_res_b+O2_moles_EGR_u+O2_moles_EGR_b
    N2_moles = N2_moles_fresh+N2_moles_res_u+N2_moles_res_b+N2_moles_EGR_u+N2_moles_EGR_b
    fuel_moles = fuel_moles_fresh + fuel_moles_res + fuel_moles_EGR
    flue_moles = flue_moles_res_b + flue_moles_EGR_b

    nT = O2_moles+N2_moles+fuel_moles+flue_moles
    fuel_conc = fuel_moles/nT
    O2_conc = O2_moles/nT
    flue_conc = flue_moles/nT

    mass_fuel = fuel_moles*mwF
    mass_air = O2_moles*(mwO2 + 3.76*mwN2)

    !equiv_ratio = AFS*mass_fuel/mass_air

    FO_ratio_u = stoic_coeff_O2*fuel_conc/O2_conc
    FO_ratio_b = FO_ratio_num/FO_ratio_den
    equiv_ratio_u = FO_ratio_u*(0.21*mwF*AFS)/(stoic_coeff_O2*mwO2)
    equiv_ratio = equiv_ratio_u

!   Compute mixture temperature FO_ratio
    energy_internal_fresh = e_given_T_p_FOR(T_fresh,pressure(1),FO_ratio_fresh,'u')
    energy_internal_res_b = e_given_T_p_FOR(T_res,pressure(1),FO_ratio_res_b,'b')
    energy_internal_res_u = e_given_T_p_FOR(T_res,pressure(1),FO_ratio_res_u,'u')
    energy_internal_res =
energy_internal_res_u*residual_fraction_u+energy_internal_res_b*residual_fraction_b
```

```
    energy_internal_EGR_b = e_given_T_p_FOR(T_EGR,pressure(1),FO_ratio_EGR_b,'b')
    energy_internal_EGR_u = e_given_T_p_FOR(T_EGR,pressure(1),FO_ratio_EGR_u,'u')
    energy_internal_EGR =
energy_internal_EGR_u*EGR_fraction_u+energy_internal_EGR_b*EGR_fraction_b
    energy_internal_u = residual_fraction*energy_internal_res+EGR_fraction*energy_internal_EGR &
                              +(1-residual_fraction-EGR_fraction)*energy_internal_fresh
    T_guess1 = Max(T_res,T_EGR)
    T_guess2 = T_fresh
    T_u = T_given_e_p_resfrac(T_guess1,T_guess2,pressure(1))
    T_b = T_u
!   R_u, R_b variations are ignored for now,
!   to be considered in the future

    R_u = R_univ/0.0286_ep
    R_b = R_univ/0.0286_ep

    rho_s = P_s/(R_u*T_s)

    V_u = volume(1)
    V_b = 0.0
    flame_radius = 0.0
    Area_f = 0.0

    cmass_u = pressure(1)*V_u/(R_u*T_u)
    cmass_b = pressure(1)*V_b/(R_b*T_b)
    cmass = cmass_u + cmass_b
    mass_ent = 0.0

    T_avg = (cmass_u*T_u + cmass_b*T_b)/cmass
    !T_wall = 450.0

    energy_internal_b = e_given_T_p_FOR(T_b,pressure(1),FO_ratio_b,'b')

    ht_rate = Heat_Transfer_Rate(1)

    ht_rate_u = ht_rate*(T_wall-T_u)/(T_wall-T_avg)
    ht_rate_u = ht_rate_u*(V_u/volume(1))**(2/3.0_ep)
    ht_rate_b = ht_rate - ht_rate_u

    Ignition = .False.
    Quenched = .False.
    Write(unit=*,fmt=*)
    Write(unit=*,fmt=*)
    Write(unit=*,fmt=*)
    Write(unit=*,fmt=*) 'Initial state :'

    Write(unit=*,fmt=*)
    Write(unit=*,fmt=*) 'masses      (kg)         : ',cmass_u,cmass_b
    Write(unit=*,fmt=*) 'Volumes    (m^3)         : ',V_u,V_b
    Write(unit=*,fmt=*) 'Temperatures (K)         : ',T_u,T_b
    Write(unit=*,fmt=*) 'Internal energies (J/kg) : ',energy_internal_u,energy_internal_b

    Write(unit=*,fmt=*) 'pressure   (atm)         : ',pressure(1)/1.01325e5
    Write(unit=*,fmt=*)
    Write(unit=*,fmt=*)
    Return
End Subroutine Initialize

*** END FILE INITIALIZE.F90 ***
```

```
*** BEGIN FILE MODEL_LOOP.F90 ***

Subroutine Model(i,outunit1,outunit2)
    Use RMap_Globals
    Use Random_Normal_Mod       !   Randlib90 random normal module

!   Variables

    Implicit None

!   outunit = unit of output file; i = current cycle
    Integer, Intent(in) :: outunit1,outunit2,i
    Real(ep) :: Q_HV

!   Noise on various parameters as specified

    Select Case (NType)
        Case (1)                                            !   Gaussian (Normal) noise on
parameters
            equiv_ratio_fresh = Random_Normal(equiv_ratio_In,equiv_ratio_Noise)
            EGR_fraction = Random_Normal(EGRIn,EGRNoise)
            residual_fraction = Random_Normal(ResIn,ResNoise)
            TurbLevel = Random_Normal(u_prime_in,u_prime_noise)
        Case Default                                        !   No noise on parameters
            equiv_ratio_fresh = equiv_ratio_In
            EGR_fraction = EGRIn
            residual_fraction = ResIn
            TurbLevel = u_prime_in
    EndSelect

    FO_ratio_fresh = stoic_coeff_O2*mwO2*equiv_ratio_fresh/(0.21*mwF*AFS)
    Write (unit=*,fmt=*)
    Write (unit=*,fmt=130) i
    Write (unit=*,fmt=140) equiv_ratio_fresh,FO_ratio_fresh

!   Initialize variables for next cycle and simulate cycle
    Call Initialize
    Write (unit=*,fmt=145) equiv_ratio_u,FO_ratio_u
    Write (unit=*,fmt=*)
    Write (unit=*,fmt=150)
    Call Cycle_Simulation(HR(i),HR2(i),outunit2)

!   Calculate Gross Heat Release (mass fuel burned * lower heating value)
    Q_HV = burn_fraction*mass_fuel*LHV

    Write (unit=outunit1,fmt=120)
i,HR(i),HR2(i),Q_HV,burn_fraction,mass_air,mass_fuel,equiv_ratio_fresh,nT
    Return
    100 Format (1x,a)
    110 Format (1x,'Warning: Total number of moles does not match. ',F8.5,',',F8.5)
    120 Format (1x,i5,8(',',E15.8E3))
    130 Format (1x,'Cycle: ',i7)
    140 Format (1x,'Input Equivalence Ratio: ',F11.8,3x,'Input Molar F/O Ratio: ',F11.8)
    145 Format (1x,'Total Equivalence Ratio: ',F11.8,3x,'Total Molar F/O Ratio: ',F11.8)
    150 Format (' Theta   burn frac (m)    burn frac (V)')
End Subroutine Model

*** END FILE MODEL_LOOP.F90 ***
```

```
*** BEGIN FILE CYCLE.F90 ***

Subroutine Cycle_Simulation(HeatRelease,HeatRelease_alt,outunit)
    Use RMap_Globals

    Implicit None

!   Additional guesses at values of variables for iteration
    Real(ep) :: T_u_guess,T_b_guess,V_u_guess,pressure_guess,vol_next
    Real(ep), Intent(Out) :: HeatRelease,HeatRelease_alt
!   ntime = index of current step; iteration = number of iterations of P,T,V, etc.
    Integer :: ntime,iteration,outunit,cycle_length,i,nstart,nstop
    Real(sp) :: const1,const2
    Real(ep) :: dpdt
    Real(ep), Dimension(:), Allocatable :: dq
!   Volume_Function is a function that determines the volume, given theta.
    Real(ep) :: Volume_Function
    Real(ep) :: Heat_Release
    Real(ep) :: SL,SL0,dmbdt,dmedt

    cycle_length = evo/dtheta - soc/dtheta + 1
    Allocate(dq(cycle_length))

    Do i = 1, cycle_length
        dq(i) = 0.0
    EndDo
    HeatRelease = 0.0
    HeatRelease_alt = 0.0
    const1 = (gamma/(gamma - 1))
    const2 = (1/(gamma - 1))
    i = 0
!   Start integration
    theta = -pi

    Loop11: Do ntime=1,nsteps       !   nsteps, dtheta, etc. defined in Input subroutine
        theta = theta + dtheta
!   Update old values
        Call Update

!   Determine new volume, combustion chamber height
        h = volume(ntime)/(0.25*pi*bore**2)
        dh = h - h_old

        piston_speed = mean_piston_speed*0.5*pi*Sin(theta)*(1+Cos(theta)/Sqrt(rod_crank_ratio**2-
(Sin(theta))**2))

!   Assuming initial turbulent intensity is TurbLevel% of piston speed at ignition
!   integral scale (size of large eddies) same as height of combustion chamber
        If (.not. Ignition) Then
            u_prime_0 = TurbLevel*Abs(piston_speed)
            li_0 = h
            rho_0 = rho_u
            P_0 = pressure(ntime)
            If (P_0 <= 0) P_0 = MAP
            T_0 = T_u
        EndIf
!   If past spark, assume ignition
        If ((theta >= soc) .and. (Motored == .False.)) Ignition = .True.
        iteration = 0
        relax_fac = 1.0

!   Initialize pressure for current step
        If (pressure(ntime) == 0) pressure(ntime) = pressure(ntime-1)
!   Start iterations with initial guess values
        Loop13: Do
            V_u_guess = V_u
            pressure_guess = pressure(ntime)
            T_b_guess = T_b
            T_u_guess = T_u
```

```
            V_b = volume(ntime) - V_u
            If (V_b < 0) Then
                V_b = 0
                V_u = volume(ntime)
            EndIf
            If (V_b > volume(ntime)) Then
                V_b = volume(ntime)*((rho_b*burn_fraction+rho_u*(1-burn_fraction))/rho_b)
                V_u = volume(ntime)-V_b
            EndIf

            iteration = iteration + 1

!   compute masses of burned and unburned states
            If (Motored) Then
                cmass_b = 0.0
            Else
                If (WiebeFcn) Then
                    Call Wiebe()
    ! Wiebe function subroutine
                Else
                    Call Burn(pressure(ntime),volume(ntime),equiv_ratio_u,SL,SL0,dmedt,dmbdt)
    ! Blizard-Keck style burning model subroutine
                EndIf
            EndIf

!   If solution is oscillating, modify the relaxation factor to achieve convergence
            If ((Abs(burn_fraction-burn_fraction_2)/burn_fraction <= epsl)) Then
                If (Abs(burn_fraction-1) < epsl) burn_fraction = 1
                If (Mod(iteration,50) == 0) Then
                    relax_fac = relax_fac-0.01
                    If (relax_fac <= eps) relax_fac = 1.0
                    cmass_b = cmass_b*1.01
                EndIf
            EndIf

!   enforce energy balance
            Call Energy(ntime)

!           If (Mod(iteration,1000) == 0) Then
!               Write (unit=*,fmt=110) Theta*180/pi,burn_fraction,v_b/volume(ntime)
!               Write (unit=*,fmt=110) Theta*180/pi,pressure(ntime)/1000.,pressure_guess/1000.
!           EndIf

            burn_fraction_2 = burn_fraction_old
            burn_fraction_old = burn_fraction

!   Check for convergence
            If(Abs(V_u-V_u_guess) > eps*volume(ntime)) Cycle Loop13
            If(Abs(pressure(ntime)-pressure_guess) > epsi*pressure(ntime)) Cycle Loop13
            If(Abs(T_u-T_u_guess) > eps*T_u) Cycle Loop13
            If(Abs(T_b-T_b_guess) > eps*T_u) Cycle Loop13
            Exit Loop13
        EndDo Loop13
        HeatRelease_alt = HeatRelease_alt + dedt
!   Calculate Heat Release (same method as used for experimental data)
!       If ((theta > soc) .and. (theta < evo)) Then
!           i = 10*(theta - soc)*180/pi + 1
!           dpdt = (pressure(ntime) - pressure(ntime - 1))/(dtheta)
!
!           dq(i) = (const1*pressure(ntime)*dvdt(ntime) + const2*volume(ntime)*dpdt)
!       EndIf

        If(Mod(ntime,100) == 0) Write(unit=*,fmt=100) theta*180/pi,burn_fraction,V_b/volume(ntime)

        If (CycleOutput) Then
            Write (unit=outunit,fmt=120)
360+theta*180/pi,T_u,T_b,pressure(ntime)*0.001,burn_fraction,V_b/volume(ntime),volume(ntime),SL,SL
0,rho_u,Area_f,u_prime,flame_radius,dmbdt,h,bore*0.5,dvdt(ntime),ht_rate,energy_internal_u,energy_
internal_b
```

```
        EndIf
    EndDo Loop11

    HeatRelease = Heat_Release()

!   nstart = (soc + pi)/dtheta + 1
!   nstop = (evo + pi)/dtheta - 1

!   Loop11b: Do ntime = nstart, nstop
!       i = ntime - nstart + 1
!       dpdt = (pressure(ntime) - pressure(ntime - 1))/(dtheta)
!       dq(i) = (const1*pressure(ntime)*dvdt(ntime) + const2*volume(ntime)*dpdt)
!   EndDo Loop11b
!   Call DataSmooth(dq, 1, cycle_length)
!   Loop11c: Do ntime = nstart, nstop - 1
!       i = ntime - nstart + 1
!
!       HeatRelease = HeatRelease + (dq(i) + dq(i + 1)) * dtheta
!   EndDo Loop11c
!   HeatRelease = 0.5*1000*HeatRelease


    100 Format (F6.1,2(2x,F15.12))
    110 Format (F6.1,2(2x,G16.8))
    120 Format (1x,F8.2,19(',',E15.8E3))
    Return
End Subroutine Cycle_Simulation

*** END FILE CYCLE.F90 ***
```

```
*** BEGIN FILE ENERGY.F90 ***

Subroutine Energy(i)
    Use RMap_Globals

    Implicit None

!   i = current cycle index - determines which entry in pressure array to use
    Integer, Intent(In) :: i
!   heat transfer rates, etc.
    Real(ep) :: dmeu,dmeb,ht_rate_u_mid,ht_rate_b_mid,p_mid,h_u_mid
    Real(ep) :: Heat_Transfer_Rate,T_given_e_p_FOR,T_given_e_p_resfrac

!   Trapezoial implicit scheme is used for integeration of the energy equations

!   Compute average tempeature and overall heat transfer rate
    T_avg = (cmass_u*T_u + cmass_b*T_b)/cmass
!   If (Motored) Then
!       ht_rate = 0.0
!   Else
        ht_rate = Heat_Transfer_Rate(i)
!   EndIf

!   Apportion the heat transfer rate among unburned & burned states
    ht_rate_u = ht_rate*(((V_u/volume(i))**2)**(1/3.0_ep))*(T_wall-T_u)/(T_wall-T_avg)
    ht_rate_b = ht_rf*(ht_rate - ht_rate_u)

    ht_rate_u_mid = 0.5*(ht_rate_u + ht_rate_u_old)
    ht_rate_b_mid = 0.5*(ht_rate_b + ht_rate_b_old)

    ht_rate_u_mid = ht_rate_u
    ht_rate_b_mid = ht_rate_b

!   compute average pressure during the crank-angle increment

    p_mid = 0.5*(pressure(i)+pressure_old(i))

!   compute the enthalpy of the unburned side

    h_u_mid = 0.5*(energy_internal_u + R_u*T_u + energy_internal_u_old + R_u*T_u_old)

!   compute specific internal energies of the 2 states

    dmeu = ht_rate_u_mid*dtheta - p_mid*(V_u-V_u_old) + (cmass_u - cmass_u_old)*h_u_mid
    If(cmass_u/cmass > epsi) energy_internal_u = (dmeu +
cmass_u_old*energy_internal_u_old)/cmass_u

    dmeb = ht_rate_b_mid*dtheta - p_mid*(V_b-V_b_old) + (cmass_b - cmass_b_old)*h_u_mid
    If(cmass_b/cmass > epsi) energy_internal_b = (dmeb +
cmass_b_old*energy_internal_b_old)/cmass_b

    dedt = (dmeu*cmass_u + dmeb*cmass_b)/cmass - ht_rate*dtheta

!   Compute temperature of the unburned side given specific internal energy, pressure, phi
    !If ((Theta*180/pi >= -151.5) .and. (Theta*180/pi <= -151.3)) Print *,T_u,T_u_old,pressure(i)
    T_u = T_given_e_p_resfrac(T_u,T_u_old,pressure(i))
    !If ((Theta*180/pi >= -151.5) .and. (Theta*180/pi <= -151.3)) Print *,T_u,T_u_old,pressure(i)
!   Compute pressure from the 2 equations of state

    pressure(i) = relax_fac*(cmass_u*R_u*T_u+cmass_b*R_b*T_b)/volume(i)+(1-relax_fac)*pressure(i)

!   Check if the pressure is within bounds

    If((pressure(i) > p_max_u) .or. (pressure(i) > p_max_b) .or. (pressure(i) < p_min_u) .or.
(pressure(i) < p_min_b)) Then
        Write(unit=*,fmt=*) 'Tu,Tb,P: ',T_u,T_b,pressure(i)
        Write(unit=*,fmt=*) 'Vu/vol,Vb/vol: ',V_u/volume(i),V_b/volume(i)
        Write(unit=*,fmt=*) 'mb,mb/m: ',cmass_b,burn_fraction
        Write(unit=*,fmt=*) 'mu,mu/m: ',cmass_u,cmass_u/cmass
```

```
        Write(unit=*,fmt=*) 'Pu-max,Pu-min: ',p_max_u,p_min_u
        Write(unit=*,fmt=*) 'Pb-max,Pb-min: ',p_max_b,p_min_b
    EndIf

!   Compute temperature of the burned side given specific internal energy, pressure, phi

    T_b = T_given_e_p_FOR(energy_internal_b,pressure(i),FO_ratio_b,'b')

    gamma_u = T_u*R_u/energy_internal_u + 1
    gamma_b = T_b*R_b/energy_internal_b + 1
    gamma = (gamma_u*cmass_u+gamma_b*cmass_b)/cmass

!   Compute unburned and burned gas volumes

    rho_b = pressure(i)/(R_b*T_b)
    rho_u = pressure(i)/(R_u*T_u)

    V_b = cmass_b/rho_b

    V_u = volume(i) - V_b

    Return
End Subroutine Energy

*** END FILE ENERGY.F90 ***
```

```
*** BEGIN FILE E_GIVEN_T_P_FOR.F90 ***

!   Computes specific internal energy for given T, P, phi

Function e_given_T_p_FOR(T,p,phi,zone)
    Use RMap_Globals

    Implicit None

!   zone = u or b, indicates whether to use burned or unburned table
    Character(Len=1), Intent(In) :: zone
!   T, P, phi = Temperature, pressure, FO_ratio to use for lookup
    Real(ep), Intent(In) :: T,p,phi
    Real(ep) :: e_given_T_p_FOR

!   e*** = pointers to table values between which to interpolate for a property
    Real(sp), Pointer :: e111,e112,e121,e122,e211,e212,e221,e222
!   Pointers to associate with the appropriate thermo table variables
    Real(sp), Pointer :: p_min,p_max,T_min,T_max,FO_ratio_min,FO_ratio_max
    Real(sp), Dimension(:), Pointer :: table_p,table_FO_ratio,table_T
    Real(sp), Dimension(:,:,:), Pointer :: thermo
    Integer, Pointer :: ntable_p,ntable_FO_ratio,ntable_T
!   interpolated values
    Real(ep) x1,x2,y1,y2,z1,z2
!   Locations in table between which to interpolate
    Integer :: i,i1,i2,ix1,ix2,iy1,iy2,iz1,iz2

!   Associate pointer variables with appropriate targets, given zone
    Select Case(zone)
        Case('u')
            ntable_p => ntable_p_u
            p_min => p_min_u
            p_max => p_max_u
            table_p => table_p_u
            ntable_FO_ratio => ntable_FO_ratio_u
            FO_ratio_min => FO_ratio_min_u
            FO_ratio_max => FO_ratio_max_u
            table_FO_ratio => table_FO_ratio_u
            ntable_T => ntable_T_u
            T_min => Tmp_min_u
            T_max => Tmp_max_u
            table_T => table_T_u
            thermo => thermo_u
        Case('b')
            ntable_p => ntable_p_b
            p_min => p_min_b
            p_max => p_max_b
            table_p => table_p_b
            ntable_FO_ratio => ntable_FO_ratio_b
            FO_ratio_min => FO_ratio_min_b
            FO_ratio_max => FO_ratio_max_b
            table_FO_ratio => table_FO_ratio_b
            ntable_T => ntable_T_b
            T_min => Tmp_min_b
            T_max => Tmp_max_b
            table_T => table_T_b
            thermo => thermo_b
    EndSelect
    !If ((Theta*180/pi >= -151.5) .and. (Theta*180/pi <= -151.3))
    !   Write (unit=*,fmt=220) 'egtpf: ',Theta*180/pi,T,P,Phi,zone

    Do i = 1,ntable_p
        table_p(i) = p_min + (p_max-p_min)*Real(i-1,ep)/Real(ntable_p-1,ep)
    EndDo

    Do i = 1,ntable_FO_ratio
        table_FO_ratio(i) = FO_ratio_min + (FO_ratio_max-FO_ratio_min)*Real(i-
1,ep)/Real(ntable_FO_ratio-1,ep)
    EndDo
```

```
Do i = 1,ntable_T
    table_T(i) = T_min + (T_max-T_min)*Real(i-1,ep)/Real(ntable_T-1,ep)
EndDo

If((p < table_p(1)) .or. (p > table_p(ntable_p))) Then
    Write (unit=*,fmt=200) 'Pressure outside the allowed range'
    Write (unit=*,fmt=200) 'in Subroutine e_given_T_P_FOR'
    Write (unit=*,fmt=210) P
    Stop
EndIf

If((T < table_T(1)) .or. (T > table_T(ntable_T))) Then
    Write (unit=*,fmt=200) 'Temperature outside the allowed range'
    Write (unit=*,fmt=200) 'in Subroutine e_given_T_P_FOR'
    Write (unit=*,fmt=210) T
    Stop
EndIf

If((phi < table_FO_ratio(1)) .or. (phi > table_FO_ratio(ntable_FO_ratio))) Then
    Write (unit=*,fmt=200) 'Equivalence ratio outside the allowed range'
    Write (unit=*,fmt=200) 'in Subroutine e_given_T_P_FOR'
    Write (unit=*,fmt=210) Phi
    Stop
EndIf

i2 = 1
Loop12: Do
    i2 = i2+1
    If(table_p(i2) <= p) Cycle Loop12
    Exit Loop12
EndDo Loop12
i1 = i2 - 1

x1 = (p-table_p(i1))/(table_p(i2)-table_p(i1))
x2 = (table_p(i2)-p)/(table_p(i2)-table_p(i1))

ix1 = i1
ix2 = i2

i2 = 1
Loop13: Do
    i2 = i2+1
    If(table_FO_ratio(i2) <= phi) Cycle Loop13
    Exit Loop13
EndDo Loop13
i1 = i2 - 1

y1 = (phi-table_FO_ratio(i1))/(table_FO_ratio(i2)-table_FO_ratio(i1))
y2 = (table_FO_ratio(i2)-phi)/(table_FO_ratio(i2)-table_FO_ratio(i1))

iy1 = i1
iy2 = i2

i2 = 1
Loop14: Do
    i2 = i2+1

    If(table_T(i2) <= T) Cycle Loop14

    Exit Loop14
EndDo Loop14
i1 = i2 - 1

z1 = (T-table_T(i1))/(table_T(i2)-table_T(i1))
z2 = (table_T(i2)-T)/(table_T(i2)-table_T(i1))

iz1 = i1
iz2 = i2
```

```
    e111 => thermo(ix1,iy1,iz1)
    e112 => thermo(ix1,iy1,iz2)
    e121 => thermo(ix1,iy2,iz1)
    e122 => thermo(ix1,iy2,iz2)

    e211 => thermo(ix2,iy1,iz1)
    e212 => thermo(ix2,iy1,iz2)
    e221 => thermo(ix2,iy2,iz1)
    e222 => thermo(ix2,iy2,iz2)

    e_given_T_p_FOR = x2*y2*z2*e111 + x2*y2*z1*e112 + x2*y1*z2*e121 + x2*y1*z1*e122 +
x1*y2*z2*e211 &
                           + x1*y2*z1*e212 + x1*y1*z2*e221 + x1*y1*z1*e222

    Return

    200 Format (a)
    210 Format (E16.8)
    220 Format (a7,4(E16.8,2x),a1)
End Function e_given_T_p_FOR

*** END FILE E_GIVEN_T_P_FOR.F90 ***
```

```
*** BEGIN FILE T_GIVEN_E_P_FOR.F90 ***

!   Computes T for given specific internal energy,P,phi

Function T_given_e_p_FOR(eint,p,phi,zone)

    Use RMap_Globals

    Implicit None

!   zone = u or b, indicates whether to use burned or unburned table
    Character(Len=1), Intent(In) :: zone
!   eint, P, phi = internal energy, pressure, FO_ratio to use for lookup
    Real(ep), Intent(In) :: eint,p,phi
    Real(ep) :: T_given_e_p_FOR
    Real(ep) :: e_lower,e_upper,eint1,eint2
    Real(sp), Pointer :: e11,e12,e21,e22
    Real(sp), Pointer :: p_min,p_max,FO_ratio_min,FO_ratio_max,T_min,T_max
    Real(sp), Dimension(:), Pointer :: table_p,table_FO_ratio,table_T
    Real(sp), Dimension(:,:,:), Pointer :: thermo
    Real(ep) :: x1,x2,y1,y2,z1,z2
    Integer, Pointer :: ntable_p,ntable_FO_ratio,ntable_T
    Integer :: i,i1,i2,ix1,ix2,iy1,iy2

!   Associate pointer variables with appropriate targets, given zone
    Select Case(zone)
        Case('u')
            ntable_p => ntable_p_u
            p_min => p_min_u
            p_max => p_max_u
            table_p => table_p_u
            ntable_FO_ratio => ntable_FO_ratio_u
            FO_ratio_min => FO_ratio_min_u
            FO_ratio_max => FO_ratio_max_u
            table_FO_ratio => table_FO_ratio_u
            ntable_T => ntable_T_u
            T_min => Tmp_min_u
            T_max => Tmp_max_u
            table_T => table_T_u
            thermo => thermo_u
        Case('b')
            ntable_p => ntable_p_b
            p_min => p_min_b
            p_max => p_max_b
            table_p => table_p_b
            ntable_FO_ratio => ntable_FO_ratio_b
            FO_ratio_min => FO_ratio_min_b
            FO_ratio_max => FO_ratio_max_b
            table_FO_ratio => table_FO_ratio_b
            ntable_T => ntable_T_b
            T_min => Tmp_min_b
            T_max => Tmp_max_b
            table_T => table_T_b
            thermo => thermo_b
    EndSelect

    Do i = 1,ntable_p
        table_p(i) = p_min + (p_max-p_min)*Real(i-1,ep)/Real(ntable_p-1,ep)
    EndDo

    Do i = 1,ntable_FO_ratio
        table_FO_ratio(i) = FO_ratio_min + (FO_ratio_max-FO_ratio_min)*Real(i-
1,ep)/Real(ntable_FO_ratio-1,ep)
    EndDo

    Do i = 1,ntable_T
        table_T(i) = T_min + (T_max-T_min)*Real(i-1,ep)/Real(ntable_T-1,ep)
    EndDo
```

```
    If((p < table_p(1)) .or. (p > table_p(ntable_p))) Then
        Write (unit=*,fmt=300) 'Pressure outside the allowed range'
        Write (unit=*,fmt=300) 'in Subroutine T_given_e_P_FOR'
        Write (unit=*,fmt=310) P
        Stop
    EndIf

    If((phi < table_FO_ratio(1)) .or. (phi > table_FO_ratio(ntable_FO_ratio))) Then
        Write (unit=*,fmt=300) 'Equivalence ratio outside the allowed range'
        Write (unit=*,fmt=300) 'in Subroutine T_given_e_P_FOR'
        Write (unit=*,fmt=310) Phi
        Stop
    EndIf

    i2 = 1
    Loop12: Do
        i2 = i2+1
        If(table_p(i2) <= p) Cycle Loop12
        Exit Loop12
    EndDo Loop12
    i1 = i2 - 1

    x1 = (p-table_p(i1))/(table_p(i2)-table_p(i1))
    x2 = (table_p(i2)-p)/(table_p(i2)-table_p(i1))

    ix1 = i1
    ix2 = i2

    i2 = 1
    Loop13: Do
        i2  = i2+1
        If(table_FO_ratio(i2) <= phi) Cycle Loop13
        Exit Loop13
    EndDo Loop13
    i1 = i2 - 1

    y1 = (phi-table_FO_ratio(i1))/(table_FO_ratio(i2)-table_FO_ratio(i1))
    y2 = (table_FO_ratio(i2) - phi)/(table_FO_ratio(i2)-table_FO_ratio(i1))

    iy1 = i1
    iy2 = i2

    e11 => thermo(ix1,iy1,1)
    e12 => thermo(ix1,iy2,1)
    e21 => thermo(ix2,iy1,1)
    e22 => thermo(ix2,iy2,1)

    e_lower = x2*y2*e11 + x2*y1*e12 + x1*y2*e21 + x1*y1*e22

    e11 => thermo(ix1,iy1,ntable_T)
    e12 => thermo(ix1,iy2,ntable_T)
    e21 => thermo(ix2,iy1,ntable_T)
    e22 => thermo(ix2,iy2,ntable_T)

    e_upper = x2*y2*e11 + x2*y1*e12 + x1*y2*e21 + x1*y1*e22

!   If((eint < e_lower) .or. (eint > e_upper)) Then
!       Write(unit=*,fmt=*) 'Internal energy outside allowable T-range'
!       Write(unit=*,fmt=*) 'Internal energy :'
!       Write(unit=*,fmt=31) eint
!       Write(unit=*,fmt=*) 'Lower and upper bounds for given P & FO_ratio'
!       Write(unit=*,fmt=32) e_lower,e_upper
!       31 Format(d24.16)
!       32 Format(2d24.16)
!       Stop
!   EndIf

    If(eint < e_lower) Then
        T_given_e_p_FOR = table_T(1)
```

```
        Return
    EndIf
    If(eint > e_upper) Then
        T_given_e_p_FOR = table_T(ntable_T)
        Return
    EndIf

    eint2 = e_lower

    i2 = 1
    Loop14: Do
        eint1 = eint2
        i2 = i2+1

        e11 => thermo(ix1,iy1,i2)
        e12 => thermo(ix1,iy2,i2)
        e21 => thermo(ix2,iy1,i2)
        e22 => thermo(ix2,iy2,i2)

        eint2 = x2*y2*e11 + x2*y1*e12 + x1*y2*e21 + x1*y1*e22

        if(eint2 <= eint) Cycle Loop14
        Exit Loop14
    EndDo Loop14
    i1 = i2 - 1

    z1 = (eint - eint1)/(eint2-eint1)
    z2 = (eint2 - eint)/(eint2-eint1)

    T_given_e_p_FOR = z1*table_T(i2)+z2*table_T(i1)

    Return
    300 Format (a)
    310 Format (E16.8)
End Function T_given_e_p_FOR

*** END FILE T_GIVEN_E_P_FOR.F90 ***
```

```
*** BEGIN FILE T_GIVEN_E_P_RESFRAC.F90 ***

Function T_given_e_p_resfrac(T_guess1,T_guess2,P)
    Use RMap_Globals

    Implicit None
    Real(ep), Intent(InOut) :: T_guess1,T_guess2
    Real(ep), Intent(In) :: P
    Real(ep) :: T_given_e_p_resfrac
    Real(ep) :: T_given_e_p_FOR,e_given_T_p_FOR

    If((residual_fraction+EGR_fraction) < 1.0e-4) Then
        T_guess1 = T_given_e_p_FOR(energy_internal_u,P,FO_ratio_fresh,'u')
        T_given_e_p_resfrac = T_guess1
        Return
    EndIf

    Loop37: Do
        T_guess2 = 0.5*(T_guess1+T_guess2)
        energy_internal_res_b = e_given_T_p_FOR(T_guess2,P,FO_ratio_res_b,'b')
        energy_internal_res_u = e_given_T_p_FOR(T_guess2,P,FO_ratio_res_u,'u')
        energy_internal_res =
energy_internal_res_u*residual_fraction_u+energy_internal_res_b*residual_fraction_b
        energy_internal_EGR_b = e_given_T_p_FOR(T_guess2,P,FO_ratio_EGR_b,'b')
        energy_internal_EGR_u = e_given_T_p_FOR(T_guess2,P,FO_ratio_EGR_u,'u')
        energy_internal_EGR =
energy_internal_EGR_u*EGR_fraction_u+energy_internal_EGR_b*EGR_fraction_b
        energy_internal_fresh = (energy_internal_u-energy_internal_res*residual_fraction-
energy_internal_EGR*EGR_fraction) &
                                        /(1-residual_fraction-EGR_fraction)

        T_guess1 = T_given_e_p_FOR(energy_internal_fresh,P,FO_ratio_fresh,'u')

        If(dabs(T_guess1-T_guess2) > 1.0e-3) Cycle Loop37
        Exit Loop37
    EndDo Loop37

    T_given_e_p_resfrac = T_guess1
    Return
End Function T_given_e_p_resfrac

*** END FILE T_GIVEN_E_P_RESFRAC.F90 ***
```

```
*** BEGIN FILE BURN.F90 ***

!   Turbulent Flame Speed/Mass Burning Rate Model

!   Primary combustion references:
!   Introduction to Internal Combustion Engines by Richard Stone
!   An Introduction to Combustion by Stephen R. Turns

Subroutine Burn(P,V,Phi,SL,SL0,dmedt,dmbdt)
    Use RMAP_Globals

    Implicit None

!   P = pressure, Phi = equivalence ratio
    Real(ep), Intent(In) :: P,V,Phi
    Real(ep), Intent(Out) :: SL,SL0,dmbdt,dmedt
    Real(ep) :: PhiM,Bm,B2

!   V_b_new = new V_b value for evaluation of flame radius
!   lm = Taylor microscale - characterizes eddy spacing; li = integral scale
!   nu = viscosity; SL = laminar flame speed; SL0 = laminar flame speed at standard T/P
!   tau = characteristic burn time for an eddy of size lm
!   dmedt = rate of mass entrainment; dmbdt = rate of mass burning
!   u_prime = turbulence intensity
    Real(ep) :: V_b_new,lm,li,nu,tau
!   flame_radius_guess = guess at flame radius for iteration; f1, f2 = functions for Newton's
Method
    Real(ep) :: flame_radius_guess,f1,f2
!   alpha, beta = exponents to determine SL from SL0, T, P
    Real(sp) :: alpha,beta

    Real(ep) :: frbtemp,dvbdr
    Real(ep) :: r_coord,r_cyl,r_coord_old,r0,z0,rmax

    Integer :: i


    Select Case (FuelType)
        Case ('G','g')
!   Constants from Stone/Stiesch for Gasoline; Bm, B2 in (m/s) (Rhodes & Keck, 1985):
            PhiM=1.21
            Bm=0.305
            B2=-0.549
        Case ('I','i')
!   Constants from Turns for Iso-Octane (Metghalchi & Keck, 1980):
            PhiM=1.13
            Bm=0.2632
            B2=-0.8472
        Case ('M','m')
!   Constants from Turns for Methanol:
            PhiM=1.11
            Bm=0.3692
            B2=-1.4051
        Case ('P','p')
!   Constants from Turns for Propane:
            PhiM=1.08
            Bm=0.3422
            B2=-1.3865
        Case ('R','r')
!   Constants from Turns for RMFD-303: (Indolene)
            PhiM=1.13
            Bm=0.2758
            B2=-0.7834
        Case ('E','e')
!   Constants from Gulder for Ethanol:
            ! Gulder uses a different correlation; since 3 parameters are required in either
case,
            !   the same variable names are used here as in other correlations for convenience;
            !   they are not appropriate for use in the SL0 correlation used for other fuels.
```

```
            PhiM=0.25 ! Eta in Gulder
            Bm=0.465 ! W in Gulder
            B2=6.34 ! Zeta in Gulder
        Case Default
            PhiM=1.13
            Bm=0.203
            B2=-0.85
    EndSelect

!   If combustion is complete, exit subroutine
    If (cmass_b_old == cmass) Then
        burn_fraction = 1
        V_b = V
        V_u = 0
        Return
    EndIf


!   Calculate new densities
    rho_b = P/(R_b*T_b)
    rho_u = P/(R_u*T_u)

!   Geometry - find flame surface area and burned/unburned volumes assuming spherical
!   flame front propogation, centered on the axis of the cylinder (simple geometry) for Ricardo
!   or cylindrical propogation from the side of the cylinder for CFR
    r_cyl = bore*0.5

    If (Ignition .and. (.not. Quenched)) Then    !   If ignition has occurred, step into flame
speed model
        V_b_new = cmass_b_old/rho_b              !   burned volume, as given by previous burned
mass & current density
        If (V_b_new <= 0) Then                   !   If no burned zone yet, create one
            T_b = 2500                           !   approximate adiabatic flame temp for
hydrocarbon fuel in air
            flame_radius = 1.5e-3                !   on the order of the size of a spark
            V_b_new = 1.414e-8                   !   approximately the volume given by the above
flame_radius
            rho_b = P/(R_b*T_b)
            cmass_b = V_b_new*rho_b
            burn_fraction = cmass_b/cmass
        Else
            flame_radius = flame_radius_old      !   If burned volume already exists, use previous
flame radius as starting guess
        EndIf

!   Select appropriate spark position for given engine
        !Select Case (EngineType)
        !   Case ('RIC','ric','Ric')             !   Top Spark - Ricardo Engine
        !       spark_offset = 0.0001
        !   Case ('CFR','cfr','Cfr')             !   Side Spark - CFR Engine
        !       spark_offset = bore*0.5
        !   Case Default                     !   Assume Ricardo if not specified
        !       spark_offset = 0.0001
        !EndSelect

!   Geometric calculations similar to Blizzard & Keck (SAE 740191)

!   Newton's Method to find new flame radius from burned volume, then new flame area from flame
radius
        FRLoop: Do
            flame_radius_guess = flame_radius
            r_coord_old = r_coord
            If (flame_radius_guess > h) Then
                r0 = Sqrt(flame_radius_guess**2 - h**2)
            Else
                r0 = 0
            EndIf
            rmax = Min(flame_radius_guess,r_cyl)
            If (flame_radius_guess > r_cyl) Then
```

```
                z0 = Sqrt(flame_radius_guess**2 - r_cyl**2)
            Else
                z0 = 0
            EndIf
            V_b = 2*pi*((flame_radius_guess**2 - r0**2)**1.5 - (flame_radius_guess**2 -
rmax**2)**1.5)/3.0
            V_b = V_b + pi*h*r0**2
            f1 = V_b - V_b_new
            f2 = 2*pi*flame_radius_guess*(Sqrt(flame_radius_guess**2 - rmax**2) -
Sqrt(flame_radius_guess**2 - r0**2))
            flame_radius = flame_radius_guess + f1/f2
            If (Abs(V_b - V_b_new)/V_b_new < 5*epsi) Exit FRLoop
            !Print *,flame_radius,flame_radius_guess,V_b,V_b_new
        EndDo FRLoop
        Area_f = flame_radius*(ATan(rmax/z0) - ATan(r0/h))

        If (cmass_b < cmass) Then            !   If combustion is not complete, determine
additional mass burned for this step

!   See chapter 8 of Stone's Introduction to Internal Combustion Engines for flame speed model
details
!   Exponents for Gasoline from Stone (Rhodes & Keck 1985)
            If ((FuelType == 'G').or.(FuelType == 'g')) Then
                alpha = 2.4 - 0.271*Phi**3.15
                beta = -0.357 + 0.14*Phi**2.77
            ElseIf ((FuelType == 'E').or.(FuelType == 'e')) Then
                alpha = 1.75
                If (Phi < 1.0) Then
                    beta = -0.17/Sqrt(Phi)
                Else
                    beta = -0.17*Sqrt(Phi)
                EndIf
            Else
!   General Exponents from Turns/Stone
                alpha = 2.18 - 0.8*(Phi-1)
                beta = -0.16 + 0.22*(Phi-1)
            EndIf

            If ((FuelType == 'E').or.(FuelType == 'e')) Then
                SL0 = Bm*Phi**PhiM*Exp(-B2*(Phi - 1.075)**2)
                SL = retard_factor*SL0*(T_u/300.0)**alpha*(P/100000.0)**Beta*(1-
2.06*flue_conc**0.77)       ! Mole fraction diluent basis
                ! Gulder's alpha/beta terms are based on T_s of 300 K and P_s of 100 kPa rather
than 25 C and 1 atm as elsewhere
            Else
                SL0 = Bm+B2*(Phi-PhiM)**2
!               SL = SL0*(T_u/T_s)**alpha*(P/P_s)**Beta*(1-2.1*flue_conc)               ! Mass
fraction diluent basis
                SL = retard_factor*SL0*(T_u/T_s)**alpha*(P/P_s)**Beta*(1-2.06*flue_conc**0.77)
    ! Mole fraction diluent basis
            EndIf

            If (SL < 0) SL = 0

            nu = (nu_s*rho_s/rho_u)*(T_u/T_s)**0.76
            li = li_0*(rho_0/rho_u)**(1/3.0_ep)
            u_prime = u_prime_0*(rho_u/rho_0)**(1/3.0_ep)
            lm = Sqrt(15*nu*li/u_prime)

            tau = lm/SL

            If ((cmass_b/cmass) >= laminar_fraction) Then
                dmedt = rho_u*Area_f*(u_prime + SL)
        ! Rate of mass entrainment
            Else
                dmedt = rho_u*Area_f*SL
            EndIf
            mass_ent = mass_ent_old+dmedt*dt
            If (mass_ent > cmass) mass_ent = cmass
```

```
            dmbdt = (mass_ent-cmass_b_old)/tau
        ! Rate of mass burning
            cmass_b = cmass_b_old + dmbdt*dt

        EndIf
    Else    !   No ignition - no change in burned mass
        cmass_b = cmass_b_old
        Return
    EndIf

    If (cmass_b > cmass) cmass_b = cmass     !   burned mass cannot be greater than total mass
    cmass_u = cmass - cmass_b
    burn_fraction = cmass_b/cmass
    !V_b = cmass_b/rho_b
    !V_u = V - V_b
    !If (Ignition) Write (unit=*,fmt=100) Theta*180/pi,burn_fraction,v_b/V

!   Quenching

    fuel_V_conc = (1-burn_fraction)*fuel_conc*nT/(V_u*1e6)
    O2_V_conc = (1-equiv_ratio_u*burn_fraction)*O2_conc*nT/(V_u*1e6)
    ahr_rate = ahr_A*Exp(-ahr_Ea/T_avg)*fuel_V_conc**ahr_m*O2_V_conc**ahr_n
    !If ((ahr_rate <= ahr_const) .and. Ignition .and. (.not. Quenched)) Then
        !Quenched = .True.
        !Print *,ahr_rate,ahr_const,Quenched
    !EndIf
    !Print *,ahr_rate,ahr_const,Quenched

    Return
    100 Format (F6.1,2(2x,G16.8))
End Subroutine Burn

*** END FILE BURN.F90 ***
```

```
*** BEGIN FILE WIEBE.F90 ***

Subroutine Wiebe()
    Use RMap_Globals

    Implicit None

    Real(ep) :: theta1,burn_parameter1,burn_parameter2

    burn_parameter1 = 5.0
    burn_parameter2 = 3.2

    theta1 = (theta-theta_0)/theta_b
    theta1 = max(theta1,0.0)
    theta1 = min(theta1,1.0)


    theta1= -burn_parameter1*(theta1**burn_parameter2)

    burn_fraction  = 1.0 - exp(theta1)

    cmass_b = burn_fraction*cmass
    cmass_u = cmass - cmass_b

    Return
End Subroutine Wiebe

*** END FILE WIEBE.F90 ***
```

```
*** BEGIN FILE HEAT_TRANSFER_RATE.F90 ***

Function heat_transfer_rate(i)
    Use RMap_Globals
    Implicit None

!   i = current crank angle - index for pressure array
    Integer, Intent(In) :: i
    Integer :: j
    Real(ep) :: heat_transfer_rate
    Real(ep) :: area_ht,ht_coeff,C1,C2,w

    area_ht = rod_crank_ratio + 1_ep - Cos(theta) - Sqrt(rod_crank_ratio**2 - Sin(theta)**2)
    area_ht = Atan(1.0)*2*(bore**2 + bore*stroke*area_ht)

!   If (Motored) Then
        ht_coeff = 2.466e-2*Sqrt(pressure(i)*T_wall/1000.0_ep)*(stroke*rpm/30.0_ep)*0.33
!   Else

!   Try Woschni correlation
!       C1 = 2.28
!       If (Ignition) Then
!           C2 = 0.00324
!       Else
!           C2 = 0.0
!       EndIf
!       j = 10*soc*180/pi + 1800
!
!       w = C1*mean_piston_speed + C2*displacement*T_0*(pressure(i)-pm(i))/(P_0*volume(j))
!
!       ht_coeff = 3.26*bore**(-0.2)*(pressure(i)/1000)**0.8*T_avg**(-0.55)*w**0.8
!   EndIf

    heat_transfer_rate = area_ht*ht_coeff*(T_wall - T_avg)

!   Print *,pressure(i),pm(i),P_0,T_wall,T_avg,ht_coeff,area_ht
    Return
End Function heat_transfer_rate

!   Heat transfer rate = h_c*area_ht*(T_wall-T_avg)

!   area_ht : heat transfer area
!   h_c     : heat transfer coefficient

!   Empirical correlation for h_c
!   h_c  ~ sqrt(pressure*T/1000)*(stroke*rpm/30)^0.33

*** END FILE HEAT_TRANSFER_RATE.F90 ***
```

```
*** BEGIN FILE VOLUME_FUNCTION.F90 ***

!   Computes volume as a funcion of crank angle

Function Volume_Function(angle)
    Use RMap_Globals
    Implicit None
    Real(ep) :: Volume_Function

!   Crank angle to use for volume calculation
    Real(ep), Intent(in) :: angle
    Real(ep) :: vol

    vol = 1+0.5*(compression_ratio-1)*(rod_crank_ratio+1-Cos(angle)-Sqrt(rod_crank_ratio**2-
(Sin(angle))**2))
    Volume_Function = vol*clearance_volume

    Return
End Function Volume_Function

*** END FILE VOLUME_FUNCTION.F90 ***
```

```
*** BEGIN FILE UPDATE.F90 ***

!   Update values at previous time step

Subroutine Update
    Use RMap_Globals

    Implicit None

    V_u_old = V_u
    V_b_old = V_b
    h_old = h

    cmass_u_old = cmass_u
    cmass_b_old = cmass_b
    mass_ent_old = mass_ent

    flame_radius_old = flame_radius

    T_u_old = T_u
    T_b_old = T_b

    energy_internal_u_old = energy_internal_u
    energy_internal_b_old = energy_internal_b

    ht_rate_u_old = ht_rate_u
    ht_rate_b_old = ht_rate_b

    pressure_old = pressure

    Return
End Subroutine Update

*** END FILE UPDATE.F90 ***
```

```
*** BEGIN FILE HR.F90 ***

Function Heat_Release()
    Use RMap_Globals

    Implicit None

    Real(ep) :: Heat_Release
!   Real(ep), Dimension(:), Intent(In) :: p,v,dv
!   Real(ep), Intent(In) :: soc,evo,gamma
    Real(ep), Dimension(nsteps) :: dq,dpdt
    Real(ep) :: const1,const2,hr0,hr1,hrsum,dth

!   Integer, Intent(In) :: nsteps
    Integer :: i,j,start,finish

!   initialize constants
    const1 = (gamma/(gamma - 1))
    const2 = (1/(gamma - 1))
    dth = dtheta*180/pi

!   increment through all cycles
!   Do i = 1,TotalCycles

!   initialize heat release summation variable
    hrsum = 0
    start = (soc + pi)*(180/pi)/dth
    finish = (evo + pi)*(180/pi)/dth
!   calculate heat release using trapezoidal rule

    Do j = start,finish
        dpdt(j) = (pressure(j+1) - pressure(j-1))/(2*dth)
        hr0 = (const1*pressure(j)*dvdt(j) + const2*volume(j)*dpdt(j))
        dq(j) = hr0
    EndDo
    finish = finish + 1
    dpdt(finish) = (pressure(finish+1) - pressure(finish-1))/(2*dth)
    hr1 = (const1*pressure(finish)*dvdt(finish) + const2*volume(finish)*dpdt(j))
    dq(finish) = hr1
!   smooth heat release curve and calculate net heat release

!   Call DataSmooth(dq,start,finish)
    finish = finish - 1
    Do j = start,finish
        hrsum = hrsum + (dq(j))*dth ! + dq(j+1))*((j+1) - j
    EndDo
    Heat_Release = hrsum !*1000*0.5
    Print *,'Cycle Heat Release (J): ',Heat_Release
    Return
!   EndDo
End Function Heat_Release

*** END FILE HR.F90 ***
```

```
*** BEGIN FILE LOGSETTINGS.F90 ***

Subroutine RMap_Log(FileName)
!   Write settings log file for simulation
    Use RMap_Globals
    Use ISO_Varying_String      !   Extract() and the Varying_String TYPE are defined here for
NAS compiler
    Implicit None
    Character(Len=50), Intent(in) :: FileName
    Integer :: outunit

    Call BKOutFile(FileName,outunit)
    Write (unit=outunit,fmt=100) Trim(Extract(FileName,1,Len_Trim(FileName)-4))
    Write (unit=outunit,fmt=135) 'Cycles = ',n
    Write (unit=outunit,fmt=130) 'RPM = ',RPM
    Write (unit=outunit,fmt=130) 'MAP = ',MAP
    Write (unit=outunit,fmt=130) 'Phi = ',equiv_ratio_in
    Write (unit=outunit,fmt=130) 'Phi Noise = ',equiv_ratio_noise
    Write (unit=outunit,fmt=130) 'EGR Fraction = ',EGRIn
    Write (unit=outunit,fmt=130) 'EGR Noise = ',EGRNoise
    Write (unit=outunit,fmt=130) 'Residual Fraction = ',ResIn
    Write (unit=outunit,fmt=130) 'Residual Noise = ',ResNoise
    Write (unit=outunit,fmt=130) 'Initial Turbulence Level = ',u_prime_in
    Write (unit=outunit,fmt=130) 'Initial Turbulence Noise = ',u_prime_noise
    Write (unit=outunit,fmt=130) 'Wall Temperature = ',T_wall
    Write (unit=outunit,fmt=130) 'Flame Speed Retardation Factor = ',retard_factor
    Select Case (NType)
        Case (1)
            Write (unit=outunit,fmt=100) 'Noise Type = Gaussian'
        Case (0)
            Write (unit=outunit,fmt=100) 'Noise Type = None'
        Case Default
            Write (unit=outunit,fmt=100) 'Noise Type = Unknown'
    EndSelect
    Write (unit=outunit,fmt=130) 'Spark Advance = ',Spark_Advance
    Select Case (EngineType)
        Case ('CFR','cfr','Cfr')
            Write (unit=outunit,fmt=100) 'Engine Geometry: CFR'
        Case ('RIC','ric','Ric')
            Write (unit=outunit,fmt=100) 'Engine Geometry: Ricardo'
        Case Default
            Write (unit=outunit,fmt=100) 'Engine Geometry Unknown - Default to CFR'
    EndSelect
    Select Case (FuelType)
        Case ('I','i')
            Write (unit=outunit,fmt=100) 'Fuel Type: Iso-Octane'
        Case ('G','g')
            Write (unit=outunit,fmt=100) 'Fuel Type: Gasoline'
        Case Default
            Write (unit=outunit,fmt=100) 'Fuel Type Unrecognized - Default to Iso-Octane'
    EndSelect

    Call BKFClose(1,outunit)

    Return
    100 Format (a)
    130 Format (a,f15.8)
    135 Format (a,i6)
End Subroutine RMap_Log

*** END FILE LOGSETTINGS.F90 ***
```

```
*** BEGIN FILE BKFILEIO.F90 ***

Subroutine BKInfile(FileIn,funit)
!   Prompt for file name and open file

!   FileIn = Input File Name; FExist = File Exists - logical
!   FChoice = File replacement menu choice; FSuccess = File operation successful
!   i = loop counter; ios = IO Status indicator

    Implicit None
    Integer, Intent(out) :: funit
    Logical :: FExist,FSuccess,FCon
    Character(len=*), Intent(inout) :: FileIn
    Integer :: i,ios

    FSuccess = .false.
    Do While (.not. FSuccess)
        Inquire (file=FileIn,exist=FExist)
        Do i=10,99
            Inquire (unit=i,opened=FCon)
            If (.not. FCon) Then
                funit=i
                Exit
            EndIf
        EndDo
        If (FExist) Then
            Open (unit=funit,file=FileIn,status='old',iostat=ios,action='read',position='rewind')
            If (ios.ne.0) Then
                Call BKIOErr(ios)
            Else
                FSuccess = .true.
            EndIf
        Else
            Write (unit=*,fmt=100) 'Input File not found; Enter new filename:'
            Read (unit=*,fmt='(a)') FileIn
        EndIf
    EndDo
    Return
    100 Format (1x,a)
End Subroutine BKInfile

Subroutine BKOutfile(FileOut,funit)
!   FileOut = Output File Name; FExist = File Exists - logical
!   FChoice = File replacement menu choice; FSuccess = File operation successful
!   i = loop counter; ios = IO Status indicator

    Implicit None
    Integer, Intent(out) :: funit
    Logical :: FExist,FSuccess,FCon
    Character(len=50), Intent(inout) :: FileOut
    Character(len=1) :: FChoice
    Integer :: i,ios

    FSuccess = .false.
    Do While (.not. FSuccess)
        Inquire (file=FileOut,exist=FExist)
        Do i=10,99
            Inquire (unit=i,opened=FCon)
            If (.not. FCon) Then
                funit=i
                Exit
            EndIf
        EndDo
        If (FExist) Then
            Do
                Write (unit=*,fmt=110) Trim(FileOut)
                Write (unit=*,fmt=100) '1. (R)eplace'
                Write (unit=*,fmt=100) '2. (A)ppend'
                Write (unit=*,fmt=100) '3. Enter (N)ew filename'
```

```
                Read (unit=*, fmt='(a)') FChoice
                Select Case (FChoice)
                    Case ('1','R','r')
                        Open
(unit=funit,file=FileOut,status='replace',iostat=ios,action='write',position='rewind')
                        If (ios.ne.0) Then
                            Call BKIOErr(ios)
                        Else
                            FSuccess = .true.
                            Exit
                        EndIf
                    Case ('2','A','a')
                        Open
(unit=funit,file=FileOut,status='old',iostat=ios,action='write',position='append')
                        If (ios /= 0) Then
                            Call BKIOErr(ios)
                        Else
                            FSuccess = .true.
                            Exit
                        EndIf
                    Case ('3','N','n')
                        Write (unit=*,fmt=100) 'Enter output filename: '
                        Read (unit=*,fmt='(a)') FileOut
                        Exit
                    Case Default
                        Cycle
                EndSelect
            EndDo
        Else
            Open
(unit=funit,file=FileOut,status='new',iostat=ios,action='write',position='rewind')
            If (ios.ne.0) Then
                Call BKIOErr(ios)
            Else
                FSuccess = .true.
            EndIf
        EndIf
    EndDo
    Return
    100 Format (1x,a)
    110 Format (1x,'File ',a,' already exists.')
End Subroutine BKOutfile

Subroutine BKFClose(n,funit)
!   Close input/output file(s)

!   funit = unit of file to close; ioerr = return error variable; ios = IO Status indicator
!   i = loop counter; n = number of files to close

    Implicit None
    Integer, Intent(in) :: n
    Integer, Dimension(n), Intent(in) :: funit
    Integer :: i,ios

    Do i=1,n
        Close (unit=funit(i),iostat=ios)
        If (ios.ne.0) Then
            Call BKIOErr(ios)
        EndIf
    EndDo
    Return
    100 Format (1x,a)
End Subroutine BKFClose

Subroutine BKIOErr(ios)
!   Return Error message for iostat variable ios - for NAS FortranPlus v2 compiler
    Use FORTRAN_IO_ERRORS

    Implicit None
```

```
    Integer, Intent(in) :: ios

    Select Case (ios)
        Case (-2:-1,1:67)
            Write (unit=*,fmt=100) ios,IOERR_MESSAGES(ios)
        Case (0)
            Write (unit=*,fmt=110)
        Case Default
            Write (unit=*,fmt=120)
    EndSelect
    Return

    100 Format (1x,'I/O Error ',i2,' - ',a)
    110 Format (1x,'I/O Operation Completed Successfully.')
    120 Format (1x,'Unknown I/O Error number: ',i2)
End Subroutine BKIOErr

*** END FILE BKFILEIO.F90 ***
```

```
*** BEGIN FILE CHAR_CONV.F90 ***

!   Take a string value that represents a numeric value, and read it into a numeric variable
!   While it would be possible to do conversions using IACHAR(), this method makes it easier
!   to handle various formatting (1, 1.0, 1.0e0, 1.0d0 could all represent the same value
!   given a Real(ep) variable for instance)

Subroutine CharToInt(InString,OutNum)
    Use Kinds, Only single, double, extended
    Implicit None
    Integer, Parameter :: sp = single, dp = double, ep = extended
    Integer, Intent(Out) :: OutNum
    Character(Len=*), Intent(In) :: InString

    Integer i,unitnum
    Logical FCon

    Do i=10,99
        Inquire (unit=i,opened=FCon)
        If (.not. FCon) Then
            unitnum=i
            Exit
        EndIf
    EndDo

    Open
(unit=unitnum,status='scratch',access='sequential',form='formatted',blank='null',action='readwrite
',position='rewind')
    Write (unit=unitnum,fmt='(a)') InString
    Rewind (unit=unitnum)
    Read (unit=unitnum,fmt=*) OutNum
    Close (unit=unitnum)
    Return
End Subroutine CharToInt

Subroutine CharToReal(InString,OutNum)
    Use Kinds, Only single, double, extended
    Implicit None
    Integer, Parameter :: sp = single, dp = double, ep = extended
    Real(sp), Intent(Out) :: OutNum
    Character(Len=*), Intent(In) :: InString

    Integer i,unitnum,ios
    Logical FCon

    Do i=10,99
        Inquire (unit=i,opened=FCon)
        If (.not. FCon) Then
            unitnum=i
            Exit
        EndIf
    EndDo

    Open
(unit=unitnum,status='scratch',access='sequential',form='formatted',blank='null',action='readwrite
',position='rewind')
    Write (unit=unitnum,fmt='(a)') InString
    Rewind (unit=unitnum)
    Read (unit=unitnum,fmt=*) OutNum
    Close (unit=unitnum)
    Return
End Subroutine CharToReal

Subroutine CharToDouble(InString,OutNum)
    Use Kinds, Only single, double, extended
    Implicit None
    Integer, Parameter :: sp = single, dp = double, ep = extended
    Real(dp), Intent(Out) :: OutNum
```

```
    Character(Len=*), Intent(In) :: InString

    Integer i,unitnum
    Logical FCon

    Do i=10,99
        Inquire (unit=i,opened=FCon)
        If (.not. FCon) Then
            unitnum=i
            Exit
        EndIf
    EndDo

    Open
(unit=unitnum,status='scratch',access='sequential',form='formatted',blank='null',action='readwrite
',position='rewind')
    Write (unit=unitnum,fmt='(a)') InString
    Rewind (unit=unitnum)
    Read (unit=unitnum,fmt=*) OutNum
    Close (unit=unitnum)
    Return
End Subroutine CharToDouble

Subroutine CharToExtended(InString,OutNum)
    Use Kinds, Only single, double, extended
    Implicit None
    Integer, Parameter :: sp = single, dp = double, ep = extended
    Real(ep), Intent(Out) :: OutNum
    Character(Len=*), Intent(In) :: InString

    Integer i,unitnum
    Logical FCon

    Do i=10,99
        Inquire (unit=i,opened=FCon)
        If (.not. FCon) Then
            unitnum=i
            Exit
        EndIf
    EndDo

    Open
(unit=unitnum,status='scratch',access='sequential',form='formatted',blank='null',action='readwrite
',position='rewind')
    Write (unit=unitnum,fmt='(a)') InString
    Rewind (unit=unitnum)
    Read (unit=unitnum,fmt=*) OutNum
    Close (unit=unitnum)
    Return
End Subroutine CharToExtended

Subroutine CharToComplex(InString,OutNum)
    Use Kinds, Only single, double, extended
    Implicit None
    Integer, Parameter :: sp = single, dp = double, ep = extended
    Complex, Intent(Out) :: OutNum
    Character(Len=*), Intent(In) :: InString

    Integer i,unitnum
    Logical Fcon

    Do i=10,99
        Inquire (unit=i,opened=FCon)
        If (.not. FCon) Then
            unitnum=i
            Exit
        EndIf
    EndDo
```

```
    Open
(unit=unitnum,status='scratch',access='sequential',form='formatted',blank='null',action='readwrite
',position='rewind')
    Write (unit=unitnum,fmt='(a)') InString
    Rewind (unit=unitnum)
    Read (unit=unitnum,fmt=*) OutNum
    Close (unit=unitnum)
    Return
End Subroutine CharToComplex

Subroutine CharToDblComplex(InString,OutNum)
    Use Kinds, Only single, double, extended
    Implicit None
    Integer, Parameter :: sp = single, dp = double, ep = extended
    Double Complex, Intent(Out) :: OutNum
    Character(Len=*), Intent(In) :: InString

    Integer i,unitnum
    Logical Fcon

    Do i=10,99
        Inquire (unit=i,opened=FCon)
        If (.not. FCon) Then
            unitnum=i
            Exit
        EndIf
    EndDo

    Open
(unit=unitnum,status='scratch',access='sequential',form='formatted',blank='null',action='readwrite
',position='rewind')
    Write (unit=unitnum,fmt='(a)') InString
    Rewind (unit=unitnum)
    Read (unit=unitnum,fmt=*) OutNum
    Close (unit=unitnum)
    Return
End Subroutine CharToDblComplex

Subroutine IntToChar(InNum,OutString)
    Use Kinds, Only single, double, extended
    Implicit None
    Integer, Parameter :: sp = single, dp = double, ep = extended
    Integer, Intent(In) :: InNum
    Character(Len=*), Intent(Out) :: OutString

    Integer i,unitnum
    Logical FCon

    Do i=10,99
        Inquire (unit=i,opened=FCon)
        If (.not. FCon) Then
            unitnum=i
            Exit
        EndIf
    EndDo

    Open
(unit=unitnum,status='scratch',access='sequential',form='formatted',blank='null',action='readwrite
',position='rewind')
    Write (unit=unitnum,fmt=*) InNum
    Rewind (unit=unitnum)
    Read (unit=unitnum,fmt='(a)') OutString
    Close (unit=unitnum)
    Return
End Subroutine IntToChar

Subroutine RealToChar(InNum,OutString)
    Use Kinds, Only single, double, extended
    Implicit None
```

```
    Integer, Parameter :: sp = single, dp = double, ep = extended
    Real(sp), Intent(In) :: InNum
    Character(Len=*), Intent(Out) :: OutString

    Integer i,unitnum
    Logical FCon

    Do i=10,99
        Inquire (unit=i,opened=FCon)
        If (.not. FCon) Then
            unitnum=i
            Exit
        EndIf
    EndDo

    Open
(unit=unitnum,status='scratch',access='sequential',form='formatted',blank='null',action='readwrite
',position='rewind')
    Write (unit=unitnum,fmt=*) InNum
    Rewind (unit=unitnum)
    Read (unit=unitnum,fmt='(a)') OutString
    Close (unit=unitnum)
    Return
End Subroutine RealToChar

Subroutine DoubleToChar(InNum,OutString)
    Use Kinds, Only single, double, extended
    Implicit None
    Integer, Parameter :: sp = single, dp = double, ep = extended
    Real(dp), Intent(In) :: InNum
    Character(Len=*), Intent(Out) :: OutString

    Integer i,unitnum
    Logical FCon

    Do i=10,99
        Inquire (unit=i,opened=FCon)
        If (.not. FCon) Then
            unitnum=i
            Exit
        EndIf
    EndDo

    Open
(unit=unitnum,status='scratch',access='sequential',form='formatted',blank='null',action='readwrite
',position='rewind')
    Write (unit=unitnum,fmt=*) InNum
    Rewind (unit=unitnum)
    Read (unit=unitnum,fmt='(a)') OutString
    Close (unit=unitnum)
    Return
End Subroutine DoubleToChar

Subroutine ExtendedToChar(InNum,OutString)
    Use Kinds, Only single, double, extended
    Implicit None
    Integer, Parameter :: sp = single, dp = double, ep = extended
    Real(ep), Intent(In) :: InNum
    Character(Len=*), Intent(Out) :: OutString

    Integer i,unitnum
    Logical FCon

    Do i=10,99
        Inquire (unit=i,opened=FCon)
        If (.not. FCon) Then
            unitnum=i
            Exit
        EndIf
```

```
    EndDo

    Open
(unit=unitnum,status='scratch',access='sequential',form='formatted',blank='null',action='readwrite
',position='rewind')
    Write (unit=unitnum,fmt=*) InNum
    Rewind (unit=unitnum)
    Read (unit=unitnum,fmt='(a)') OutString
    Close (unit=unitnum)
    Return
End Subroutine ExtendedToChar

Subroutine ComplexToChar(InNum,OutString)
    Use Kinds, Only single, double, extended
    Implicit None
    Integer, Parameter :: sp = single, dp = double, ep = extended
    Complex, Intent(In) :: InNum
    Character(Len=*), Intent(Out) :: OutString

    Integer i,unitnum
    Logical FCon

    Do i=10,99
        Inquire (unit=i,opened=FCon)
        If (.not. FCon) Then
            unitnum=i
            Exit
        EndIf
    EndDo

    Open
(unit=unitnum,status='scratch',access='sequential',form='formatted',blank='null',action='readwrite
',position='rewind')
    Write (unit=unitnum,fmt=*) InNum
    Rewind (unit=unitnum)
    Read (unit=unitnum,fmt='(a)') OutString
    Close (unit=unitnum)
    Return
End Subroutine ComplexToChar

Subroutine DblComplexToChar(InNum,OutString)
    Use Kinds, Only single, double, extended
    Implicit None
    Integer, Parameter :: sp = single, dp = double, ep = extended
    Double Complex, Intent(In) :: InNum
    Character(Len=*), Intent(Out) :: OutString

    Integer i,unitnum
    Logical FCon

    Do i=10,99
        Inquire (unit=i,opened=FCon)
        If (.not. FCon) Then
            unitnum=i
            Exit
        EndIf
    EndDo

    Open
(unit=unitnum,status='scratch',access='sequential',form='formatted',blank='null',action='readwrite
',position='rewind')
    Write (unit=unitnum,fmt=*) InNum
    Rewind (unit=unitnum)
    Read (unit=unitnum,fmt='(a)') OutString
    Close (unit=unitnum)
    Return
End Subroutine DblComplexToChar

*** END FILE CHAR_CONV.F90 ***
```

BIBLIOGRAPHY

1. *Concepts for improved fuel economy from gasoline engines.* **Turner, J.W.G., Pearson, R.J. and and Kenchington, S.A.** 2005, International Journal for Engine Research 6, No. 2, pp. 137-157.
2. *Analysis of stratified EGR and air on combustion and NO formation in a spark-ignition engine.* **Zhao, H., et al.** 1999, Proceedings of the Institution of Mechanical Engineers, Vol. 213 Part D, pp. 611-623.
3. *Engine performance and emissions near the dilute limit with hydrogen enrichment using an on-board reforming strategy.* **Auader, Ather A., Kirwan, John E. and Grieve, M. James.** 2003, SAE Paper 2003-01-1356.
4. *Stoichiometric operation of a gas engine utilizing synthesis gas and EGR for NOx control.* **Smith, Jack A. and Bartley, Gordon J. J.** s.l. : ASME, October 2000, Journal of Engineering for Gas Turbines and Power, Vol. 122, pp. 617-623.
5. *Effects of injection timing on the lean misfire limit in an SI engine.* **Ohm, I. Y., Jeong, K. S. and Jeung, I. S.** 0997, SAE Paper 970028.
6. *Improving NOx and fuel economy for mixture injected SI engine with EGR.* **Tabata, Michihiko, Yamamoto, Toshihide and Fukube, Tugio.** 1995, SAE Paper 950684.
7. *High efficiency stoichiometric spark ignition engines.* **De Petris, C., et al.** 1994, SAE Paper 941933.
8. *Knock behavior of a lean-burn, H2 and CO enhanced, SI gasoline engine concept.* **Topinka, Jennifer A., et al.** 2004, SAE Paper 2004-01-0975.
9. **Heywood, J.B.** *Internal Combustion Engine Fundamentals.* New York : Mcgraw-Hill, 1988.
10. **Southwest Research Institute.** SwRI: HEDGE, High Efficiency Dilute Gasoline Engine Consortium. [Online] http://www.swri.edu/4org/d03/engres/pwrtrn/hedge/default.htm.
11. *What limits lean operation in spark ignition engines - flame initiation or propagation?* **Quader, A.A.** 1976, SAE Paper 760760.
12. *The effect of spark power on spark-ignited flame kernel growth.* **Cho, Y. S., Santavicca, D. A. and Sonntag, R. M.** 1992, SAE Paper 922168.

13. *The effect of incomplete fuel-air mixing on spark-ignited flame kernel growth.* **Cho, Y. S. and Santavicca, D. A.** 1993, SAE Paper 932715.
14. **Santavicca, Domenic A.** *Spark ignited turbulent flame kernel growth.* Penn State University. University Park, PA : s.n., 1995. Prepared for the United States Department of Energy under contract no. DE-FG04-90AL62353.
15. *Investigations of the influence of mixture preparation on cyclic variations in SI-engine, using laser induced fluorescence.* **Johansson, Bengt, et al.** 1995, SAE Paper 950108.
16. *Flame chemiluminescence studies of cyclic combustion variations and air-to-fuel ratio of the reacting mixture in a lean-burn stratified-charge spark-ignition engine.* **Aleiferis, P. G., et al.** 2004, Combustion and Flame, Vol. 136, pp. 72-90.
17. *Variation of turbulent burning rate of methane, methanol, and iso-octane air mixtures with equivalence ratio at elevated pressure.* **Lawes, M., et al.** s.l. : Taylor & Francis, Inc., 2005, Combustion Science and Technology, Vol. 177, pp. 1273-1289. ISSN: 0010-2202.
18. *Comparing lean burn and EGR.* **Lumsden, Grant, Eddleston, David and Sykes, Richard.** 1997, SAE Paper 970505.
19. *Effect of combustion air dilution by water vapor or nitrogen on NOx emission in a premixed turbulent natural gas flame: an experimental study.* **Landman, M. J., Derksen, M. A. F. and Kok, J. B. W.** 2006, Combustion Science and Technology, Vol. 178, pp. 623-634. ISSN: 0010-2202.
20. *An experimental study of the cyclic variability in spark ignition engines.* **Ozdor, Nir, Dulger, Mark and Sher, Eran.** 1996, SAE Paper 960611.
21. *An examination of the role of residual gases in the combustion processes of motored engines fuelled with gaseous fuels.* **Liu, Z. and Karim, G. A.** 1996, SAE Paper 961081.
22. *Planar laser-induced fluorescence of H2O to study the influence of residual gases on cycle-to-cycle variations in SI engines.* **Juhlin, G., et al.** 1998, Combustion Science and Technology, Vol. 132, pp. 75-97.
23. *A dynamical instability of spark-ignited engines.* **Kantor, J.** 1984, Science, Vol. 224, pp. 1233-1235.

24. **Finney, Charles Edward Andrew.** *Identification and Characterization of Determinism in Spark-Ignition Internal Combustion Engines.* s.l. : Master's Thesis, University of Tennessee - Knoxville, 1995.
25. *Origins of cyclic dispersion patterns in spark ignition engines.* **Wagner, R. M., Drallmeier, J. A. and Daw, C. S.** 1998. Proceedings of the 1998 Technical Meeting of the Central States Section of the Combustion Institute.
26. *A simple model for cyclic variations in a spark-ignition engine.* **Daw, C. Stuart, et al.** 1996, SAE Paper 962086.
27. *Observing and modeling nonlinear dynamics in an internal combustion engine.* **Daw, C. S., et al.** 3, s.l. : The American Physical Society, March 1998, Physical Review E, Vol. 57, pp. 2811-2819.
28. *Prior-cycle effects in lean spark-ignition combustion - fuel/air charge considerations.* **Wagner, R. M., Drallmeier, J. A. and Daw, C. S.** 1998, SAE Paper 980147.
29. *Characterization of lean combustion instability in premixed charge spark ignition engines.* **Wagner, R. M., Drallmeier, J. A. and Daw, C. S.** 4, s.l. : Institution of Mechanical Engineers, 2000, International Journal of Engine Research, Vol. 1, pp. 301-320.
30. **van Goor, Nicolaas Aaldert.** *Control of Chaotic and Nonlinear Systems Influenced by Dynamic Noise.* s.l. : The University of Tennessee, Knoxville, PhD Dissertation, 1998.
31. *Controlling chaos.* **Ott, E., Grebogi, C. and Yorke, J. A.** 11, 1990, Physical Review Letters, Vol. 64, pp. 1196-1199.
32. *Application of nonlinear feedback control to enhance the performance of a pulsed combustor.* **Edwards, K. D., et al.** Indianapolis, Indiana : Combustion Institute, 2000. Proceedings of the 2000 Spring Technical Meeting of the Central States Section of the Combustion Institute.
33. *Controlling cyclic combustion variations in lean-fueled spark-ignition engines.* **Davis, L. I. Jr., et al.** 2001, SAE Paper 2001-01-0257.
34. **Davis, Jr. et al.** *Method of controlling cyclic variation in engine combustion. 5,921,221* United States of America, July 13, 1999.

35. *Model based control of cyclic dispersion in lean spark ignition combustion.* **Green, J. B. Jr., Wagner, R. M. and Daw, C. S.** s.l. : Combustion Institute, 2002. Proceedings of the 2002 Technical Meeting of the Central States Section of the Combustion Institute.
36. *Symbolic time-series analysis of engine combustion measurements.* **Finney, C. E. A., Green, J. B., Jr. and Daw, C. S.** 1998, SAE Paper 980624.
37. *Symbol sequence statistics in noisy chaotic signal reconstruction.* **Tang, X.Z., et al.** 1995, Physical Review E51, pp. 3871-3389.
38. *Low-order map approximations of lean cyclic dispersion in premixed spark ignition engines.* **Wagner, R. M., Daw, C. S. and Green, J. B. Jr.** 2001, SAE Paper 2001-01-3559.
39. *Artificial intelligence for the modeling and control of combustion processes: a review.* **Kalogirou, S.A.** 2003, Progress in Energy and Combustion Science, Vol. 29, pp. 515-566.
40. *Lean combustion stability of spark ignition engines using backstepping scheme.* **He, Pingan and Sarangapani, Jagannathan.** 2003. Proceedings of the IEEE Conference on Controls Applications. Vol. 1, pp. 167-172.
41. *Neuro emission controller for minimizing cyclic dispersion in spark ignition engines.* **He, Pingan and Sarangapani, Jagannathan.** 2003. Proceedings of the IEEE Joint Conference on Neural Networks. Vol. 2, pp. 1535-1540.
42. **Tunestal, Per Anders.** *The Use of Cylinder Pressure for Estimation of the In-Cylinder Air/Fuel Ratio of an Internal Combustion Engine.* s.l. : University of California, Berkeley, PhD Dissertation, 2000.
43. *Neural network-based output feedback controller for lean operation of spark ignition engines.* **Vance, Jonathan B., et al.** Minneapolis, MN : s.n., 2006. Proceedings of the American Controls Conference.
44. *Neural network control of spark ignition engines with high EGR levels.* **Singh, Atmika, et al.** Vancouver, BC, Canada : s.n., 2006. Proceedings of the IEEE World Conference on Computational Intelligence.

45. *Development and implementation of neural network controller for spark ignition engines with high EGR levels.* **Vance, J., et al.** 4, July 2007, IEEE Transactions on Neural Networks, Vol. 18.
46. *Development and implementation of neural network controller for spark ignition engines operating lean.* **Vance, J., et al.** 2008, IEEE Transactions on Control Systems Technology, to appear in 2008.
47. *A new combustion pressure sensor for advanced engine management.* **Herden, Werner and Küsell, Matthias.** 1994, SAE Paper 940379.
48. *A new design of optical in-cylinder pressure sensor for automotive applications.* **Fitzpatrick, Michael, Pechstedt, Ralf and Lu, Yicheng.** 2000, SAE Paper 2000-01-0539.
49. *Cylinder-pressure-based engine control using pressure-ratio-management and low-cost non-intrusive cylinder pressure sensors.* **Sellnau, Mark C., et al.** 2001, SAE Paper 2001--01-0932.
50. *A model for the estimation of inducted air mass and the residual gas fraction using cylinder pressure measurements.* **Mladek, Michael and Onder, Christopher H.** 2000, SAE Paper 2000-01-0958.
51. *Combustion pressure based engine management system.* **Müller, Rainer, et al.** 2000, SAE Paper 2000-01-0928.
52. *KIVA: A comprehensive model for 2D and 3D engine simulations.* **Amsden, A. A., et al.** 1985, SAE Paper 850554.
53. **Los Alamos National Laboratory.** Computational Fluid Dynamics Group. *Los Alamos National Laboratory.* [Online] http://www.lanl.gov/orgs/t/t3/codes/kiva.shtml.
54. *The KIVA-II computer program for transient multidimensional chemically reactive flows with sprays.* **Amsden, A. A., Butler, T. D. and O'Rourke, P. J.** 1987, SAE Paper 872072.
55. *KIVA-3: An unstructured KIVA program for complex geometries.* **Amsden, A. A., O'Rourke, P. J. and Butler, T. D.** Detroit, MI : Los Alamos National Laboratory LA-UR-91-138, 1991. First International KIVA Users Meeting.

56. *Coarse mesh CFD: trend analysis in a fraction of the time.* **He, Y., et al.** Detroit, MI : s.n., 2001. Abstracts of the Eleventh International Multidimensional Engine Modeling User's Group Meeting at the SAE Congress.
57. **Yaşar, Osman.** Computational Engine Modeling. *Oak Ridge National Laboratory.* [Online] http://www.ornl.gov/info/ornlreview/v30n3-4/engine.htm.
58. *A review of investigations using the second law of thermodynamics to study internal-combustion engines.* **Caton, J.A.** 2000, SAE Paper 2000-01-1081.
59. *A cycle simulation including the second law of thermodynamics for a spark-ignition engine: implications of the use of multiple-zones for combustion.* **Caton, J.A.** 2002, SAE Paper 2002-01-0007.
60. *Modeling nonlinear dynamics in a spark-ignition engine with a two-zone thermodynamic model.* **Chakravarty, Kalyan, et al.** Philadelphia, PA : s.n., March 20-23, 2005. Proceedings of the Fourth Joint Meeting of the U.S. Sections of the Combustion Institute.
61. *Experimental and theoretical investigation of turbulent burning model for internal combustion engines.* **Blizard, Norman C. and Keck, James C.** 1974, SAE Paper 740191.
62. *A turbulent entrainment model for spark-ignition engine combustion.* **Tabaczynski, R.J., Ferguson, C.R. and Radhakrishnan, K.** 1977, SAE Paper 770647.
63. *Further refinement and validation of a turbulent flame propagation model for spark-ignition engines.* **Tabaczynski, R.J., Trinker, F.H. and Shannon, B.A.S.** 1980, Combustion and Flame, Vol. 39, pp. 111-121.
64. **Stone, Richard.** *Introduction to Internal Combustion Engines.* s.l. : SAE, 1999.
65. *Turbulent flame structure and speed in spark-ignition engines.* **Keck, James C.** s.l. : Combustion Institute, 1982. Proceedings of the Nineteenth Symposium (International) on Combustion. pp. 1451-1466.
66. *Numerical simulation of combustion in premixed SI engines using fractal flame models.* **De Petris, C., et al.** 1995, SAE Paper 952383.
67. *Chemical closure model for fractal flamelets.* **Gouldin, F. C., Bray, K. N. C. and Chen, J. Y.** 1989, Combustion and Flame, Vol. 77, pp. 241-259.

68. *New data on flame behavior in lean burn S.I. engine.* **Pajot, O., Mounaïm-Rousselle, C. and Queiros-Conde, D.** 2001, SAE Paper 2001-01-1956.
69. *Burning velocities of air with methanol, isooctane, and indolene at high pressure and temperature.* **Metghalchi, M. and Keck, J.C.** 1982, Combustion and Flame, Vol. 48, pp. 191-210.
70. **Turns, Stephen R.** *An Introduction to Combustion.* s.l. : McGraw-Hill, 2000.
71. *Correlations of laminar combustion data for alternative S.I. engine fuels.* **Gülder, Ömer L.** 1984, SAE Paper 841000.
72. *Burning velocities of ethanol-isooctane blends.* **Gülder, Ömer L.** s.l. : Combustion Institute, 1984, Combustion and Flame, Vol. 56, pp. 261-268.
73. *A hybrid 2-zone/WAVE engine combustion model for simulating combustion instabilities during dilute operation.* **Edwards, K. Dean, et al.** 2005, SAE Paper 2005-01-3801.
74. **Evers, Matthew Ryan.** *Investigating Fuel Film Atomization From Valve and Port surfaces in a Spark Ignition Engine.* Rolla, MO : University of Missouri - Rolla, Masters Thesis, 1999.
75. **Wagner, Robert Milton.** *Identification and Characterization of Complex Dynamic Structure in Spark Ignition Engines.* s.l. : PhD Dissertation, University of Missouri - Rolla, 1999.
76. *Development of nonlinear cyclic dispersion in spark ignition engines under the influence of high levels of EGR.* **Sutton, Robert Wesley and Drallmeier, James A.** 2000. Proceedings of the 2000 Technical meeting of the Central States Section of the Combustion Institute.
77. **Sutton, Robert Wesley.** *Investigation of Cyclic Dispersion Under Lean Fueling and High Levels of Simulated EGR.* s.l. : Masters Thesis, University of Missouri - Rolla, 2000.
78. **Interactive Software Services, Ltd.** F2KCLI. October 2005.
79. *Extensions of Forsythe's method for random sampling from the normal distribution.* **Ahrens, J. H. and Dieter, U.** 1973, Math. Comput., Vol. 27, pp. 927-937. Algorithm FL (method=5).

VITA

Brian Christopher Kaul was born December 9, 1978 in St. Louis County, Missouri. After living for a short time in Charlotte, North Carolina as a child, his family moved back to the St. Louis area, where he graduated from Francis Howell North High School in 1997. He went on to study Mechanical Engineering at the University of Missouri – Rolla, where he graduated Summa Cum Laude with his Bachelor of Science degree in 2001, and went on to earn his Master of Science degree in Mechanical Engineering in 2003.

Upon completing his Master of Science degree, he continued his graduate studies under Dr. James Drallmeier in the Internal Combustion Engine Laboratory at the University of Missouri – Rolla. He earned his Doctor of Philosophy in Mechanical Engineering degree from the newly renamed Missouri University of Science and Technology in 2008.