Embedded Software Control of a Hybrid Vehicle’s ECU/FCM System

Robert Murphy and Patrick Otoo Bobbie
Department of Computer Science, School of Computing and Software Engineering, Southern Polytechnic State University, Marietta, Georgia, USA

Received: September 08, 2011 / Accepted: September 29, 2011 / Published: January 25, 2012.

Abstract: This paper describes a research project that uses embedded systems design principles to construct and simulate an Engine Control Unit (ECU) for a hybrid car. The ECU is designed to select a fuel type based on the stress level of the simulated engine. The primary goal of the project was to use a robotics kit, connected to sensors, to simulate a hybrid car under certain stress conditions such as hill climbing or full throttle. The project uses the LEGO® Mindstorms® NXT robotics kit combined with a Java-based firmware, a pressure sensor to simulate a gas pedal, and a tilt sensor to determine when the car is traveling uphill or downhill. The objective was to develop, through simulation, a framework for adjusting the ratios/proportions of fuel types and mixture under the stress conditions. The expected result was to establish a basis for determining the ideal/optimal fuel-mix-stress ratios on the hybrid car’s performance. Using the NXT robotics kit abstracted the low level details of the embedded system design, which allowed a focus on the high level design details of the research. Also, using the NXJ Java-based firmware allowed the incorporation of object oriented design principles into the project. The paper outlines the evolution and the compromises made in the choice of hardware and software components, and describes the computations and methodologies used in the project.

Key words: LEGO NXT™, embedded systems, engine control unit, NXJ Java firmware, fuel control module.

1. Introduction

In recent times, the need for alternative energy to purely petroleum-based products has spurred research at governmental, industrial, and academic levels. Most important is the need for experimental work based on simulations of multiple-fuel mixtures, as alternatives, for conventional vehicles without significant hardware modifications. Any significant progress in this regard depends largely on the quantity of alcohols (e.g., ethanol and methanol) produced by agricultural industries, considering the competing forces from human needs for corn and similar grains. Current research efforts suggest that the future growth of the ethanol industry depends on whether higher ethanol blends above E-10 are permitted in non-flex-fuel vehicles [1]. The need for higher ethanol blends above E-10 for conventional vehicles is important as well as the current policy debates aimed at encouraging alternative energy usage. For non-vehicular, or non-road, engines ethanol blends are not standardized and remain indistinguishable when the levels are set between E-10 and E-0 [1]. Thus, simulation studies that test E-10 to E-0 ranges of non-flex-fuel is needed to determine the optimal ethanol-gasoline ratios at both the E-10 to E-25 and E-10 to E-0 levels.

In an ongoing research, ethanol is increasingly being used as alternative fuel compared with the conventional petroleum-based products. While in the US, for example, ethanol-gasoline mixture is at the E-10 level, higher ratios of up to E-85 and E-93 (e.g., in Brazil) with minimum hardware modification are being used [2]. For ethanol-diesel mixtures, ethanol is often
blended with diesel fuel to produce diesohol; the goal is to achieve a lean air-to-fuel mixture in order to produce more efficient combustion. Research conducted in Ref. [3] resulted in a new hydrated-ethanol fuel—EM60 (60% ethanol and 40% H2O) which is added to diesel fuel to power diesel engines. The benefits of the EM60-based approach, according to Ref. [3], include the reduction in heat emission and friction for long engine life and provision of long storage life and stability of the fuel mixture for extreme, stressful weather conditions (e.g., on a slippery or rocky hill).

In a study of using multiple fuel mixtures to operate automobile and light trucks, special sensors were used to determine the fuel blend. Using the sensors also allowed automatic adjustment of the synchronization of ignition and air-fuel coefficients [4]. In this study [4], the ethanol-diesel blend, also known as e-diesel, was obtained by mixing bioethanol with conventional diesel oil at E-5 to E-15 levels, with an additive for stability of the blend. The study concluded that, in comparison with regular diesel fuel, e-diesel can improve cold start-up properties of diesel-only fuel and also reduce environmentally sensitivity matters. In the USA and Brazil, e-diesel has been tested while current testing is ongoing in Spain and other parts of Europe. In parts of Canada, e-diesel blends in the ratio of E-91.5 (gasoline), E-7.5 (ethanol), and E-1 (additive—co-solvent) are being tested. Although this mixture is not available commercially, it is an alternative fuel which is different from biodiesel which promises to reduce air pollutants and is aimed at large fleet usage [5].

Several companies have designed sensor-controlled instruments for measuring the air-fuel, or stoichiometric, ratios for several fuel mixtures. For gasoline engines the theoretical ratio is 14.7, 14.6 for diesel, 17.2 for CNG (natural gas), 15.5 for LPG (propane), 6.4 for methanol, and 9.0 for ethanol. When an engine runs at the lean ratio, free O2 is usually present in the exhaust with no unburned fuel. However, studies show that maximum power or engine thrust is achieved when it runs slightly at the rich ratio, i.e., with 80%-90% combustion. For single engines, such measurements are often difficult, because outside air invades the exhaust [6].

Simulation studies of e-diesel mixture in single engines will offer insights into the degree of difficulty of such measures and the advantages of using (regulated) diesel-based mix—reduced heat emission and long engine life. The research reported in this paper is focused on simulation studies of a Fuel-Control Module (FCM) of an Engine Control Unit (ECU) using sensors and the NXT™ robotic/NXJ-Java™ platform. Adopting a hardware-software co-design platform facilitated the design, implementation, data collection, and analysis of the project. Data were collected on a number of fuel-mix scenarios for a virtual car performing at different levels of maneuvers or stress. The fuel types used in the simulation were natural gas (CNG) and diesel in various mixtures.

The motivation of the Engine Control Unit project lies in a fuel control system built by Magneti Marelli, nicknamed the Omnivorous Engine. Marelli’s Brazilian engineering team produced a hybrid car that runs on both natural gas and a mixture of diesel fuel and ethanol. For the hybrid car, a driver can mix the gasoline and ethanol in any ratio, and the Engine Control Unit determines experimentally what mixture of gasoline and ethanol has been poured into the tank. It then adjusts the spark timing and air-fuel ratio in an effort to achieve total combustion. Traveling at a constant speed, the car uses natural gas, but when going uphill or passing another car the ECU signals the engine to use the gasoline and ethanol mixture [7]. This project aims to simulate the Marelli ECU and also extend the functionality to diesel engines by keeping diesel fuel and ethanol in separate tanks and mixing them on the fly. Since the sensors sense the fuel mixture being sprayed into the engine, no combustion cycles are wasted trying to determine the ratio, thus achieving consistency of the engine’s performance.

The rest of the paper is organized as follows: Section 2 is on the hardware components, including the
necessary sensors used in the mechanical design of the virtual car using an NXT™ robot instead of a conventional MC-8051 microcontroller; section 3 is on the design and integration of the system components, including the ECU and the FCM; section 4 is on data collection and analysis; and section 5 is on the assumptions made and limitations of the system; finally, the paper concludes with the observations in section 6.

2. Hardware Components

2.1 Original Component: MC-8051 Microcontroller

At the onset of the research project, the 8051 microcontroller was chosen as the hardware development platform. The 8051 development kit from Silicon Laboratories provided a microcontroller with sufficient Random Access Memory (RAM) and programmable FLASH memory for the project. The 8051 contains a powerful Interface Development Environment (IDE), reminiscent of Visual Studio 6, which allowed cross compilation from C to 8051 assembly language. The IDE also acts as a front end to the Joint Test Action Group (JTAG) debugging interface and makes it easy to step through code, examine register values, and generally test the software. The drawback to using the 8051 microcontroller in this project is the attention that has been given to low level details. Any sensors connected to the 8051 I/O pins required circuits to be built, driver software to be written, and calibration done. The bulk of the project would have been spent working on these low level aspects instead of high level design and coding to reach the project goal. For this reason, a platform conducive to rapid development that abstracts the low level hardware details was desirable. The LEGO® MINDSTORMS® NXT robotic kit was therefore chosen for this project since the NXT has natural interfaces for attaching sensors without writing new device drivers.

2.2 Final Component: The LEGO™ NXT™

The LEGO NXT platform was also chosen for two main reasons: availability of multiple programming environments such as Java, C#, C, and Ruby to work with, and the availability of both LEGO-specific and third-party sensors for the NXT. The programming environment chosen was Java, which bolsters object oriented development. The LEGO NXJ Java Virtual Machine firmware laid the groundwork for development using an Eclipse plug-in and several Java-based LEGO NXT communication utilities. The sensors were attached to the LEGO NXT simply by an RJ12 connector that looks much like a telephone cable. Once the sensors were connected and initialized by NXJ, they were polled using simple function/method calls. In contrast, the 8051-based effort would have required the construction of a circuit board and development of requisite device drivers and sensor interfaces to obtain normalized data values from the sensors.

2.3 Sensors

Two sensors provided the input needed for a meaningful simulation. A pressure sensor provided a value that represented the throttle position and simulated a driver pressing on a gas pedal. The pressure sensor works by measuring resistance along a path; the shorter the path, the less resistance. Pushing the blue plunger shortened the path to the conducting agent which gave us the pressure value. Measurements made by the sensor ranged in values from 1 Newton to 15 Newtons, which represented closed throttle and fully open throttle, respectively [8]. Fig. 1 is a photo-shot of the pressure sensor from Techno-Stuff™ that was used in the simulation.

Fig. 1 Pressure sensor.
An accelerometer was used to gather tilt information. In the simulation, the angle at which the simulated car was tilted determined whether the car was traveling uphill, downhill, or on a flat road. The accelerometer allowed the simulation of the relationship between uphill/downhill in terms of gain in velocity when traveling uphill to overcome the loss of torque; the converse was true when traveling downhill. The experimental design and its successful simulation allowed the signaling of the FCM to assist with the fuel calculations. The sensor measured values in the x-axis with the range of 0-254 degrees. Fig. 2 is photo-shot of the accelerometer from Hitechnic™ that was used in the project [9].

3. Design

As the world becomes more environmentally conscious and sensitive, more and more total electric and electric-gas hybrid vehicles would appear on the roads. This type of alternate energy vehicle design is well suited for the everyday commuter, but for commercial drivers, electric and electric hybrid vehicles are impractical and would need extremely large electrical sources to support them. For this reason, the research project was designed to simulate a single cylinder, diesel engine as opposed to a pure gasoline engine. Diesel engines use a four stroke combustion cycle: intake stroke, compression stroke, combustion stroke, and exhaust stroke. This lends itself to a state-based model using a finite state machine. Each stroke gets represented in code, and the position of the piston acts as an indicator of the state. The high level design involves several subsystems: Driver and Normalization, the ECU, the FCM, and the Engine. The system receives signals from the input sensors and normalizes them via the Driver subsystem. The ECU takes the normalized values from the Driver and makes decisions on how to control the engine based on the calculations made by the FCM.

The overall system design architecture is depicted in Figs. 3-4. The figures describe the signal data flow and control (block/unblock) states which synchronize the ECU subsystems.

3.1 Driver and Normalization Subsystem

The Driver subsystem was implemented using the LeJOS™ NXJ Application Programming Interface (API). Each of the sensor objects for interfacing with the physical sensors was instantiated from a base Sensor class. Typically, the getReading method of each Sensor class obtains a normalized reading from the
physical sensor. Since we used third party (non-LEGO) sensors, they needed to be programmed in such a way that allowed the interfacing with the NXT. For our pressure sensor (which was used as an accelerator pedal), the getReading method returned a normalized value from 1 to 100; with 1 representing idle speed. The third party sensor needed to be programmed also as a standard LEGO NXT light sensor. Operationally, the Driver subsystem polled the sensors on every combustion cycle and sent the values to the ECU. After the ECU was done with its calculations, the Driver sent command signals to the engine, informing the Engine of what fuel types to use, and in what proportions of mixture as well as when to spark the fuel mix [10].

3.2 The ECU

The general goal of the ECU was to coordinate the effort between the FCM and the Engine. Operationally, the ECU outputted fuel levels, mixtures, timing of the spark, and air-fuel ratio data into a log file. In this simulation, the ECU also functioned as a Time Control Unit (TCU). The ECU used a wait function to control timing. During execution of the wait function, the ECU looped until the crank was at the correct angle, per the FCM determination. The ECU then sent the command signals, through the Driver, to the Engine [11]. The ECU also maintained a data structure for all information related to the Engine, including ethanol, diesel fuel, natural gas levels, Engine stress, revolutions per minute of the crankshaft, and on/off status of the Engine. The objective of keeping all the performance data about the engine in a single data structure was to unify the interface to the dynamic data (of the simulation) and to support the feedback control loop design of the system. The ECU, therefore, was used as the control unit/center for this protocol of the simulation.

3.3 The FCM

The Fuel Control Module was the crux of the entire project. All decisions on spark timing, fuel type, fuel mixtures, air-fuel ratios, and Engine stress levels were made in FCM class of the simulator. The basis for all these decisions was on the Engine stress factor. Normally, an engine stress is a measure of how much stress is being put on the Engine at any given time, and is a function of the tilt of a car (uphill or downhill) and the throttle position. Specifically, Engine stress is calculated by the system as the addition of the tilt and throttle sensor values. During times of low Engine stress, the FCM selected natural gas; during times of higher engine stress, the FCM selected a mixture of ethanol and diesel fuel on a scale (proportions) of 0%-diesel over 100%-ethanol to 100%-diesel over 0%-ethanol. The fuel mixture was determined on-the-fly by the FCM based on a scale from the Engine stress. The system also compensated for fuel tanks being empty. If the ethanol tank was empty, the system only used natural gas and diesel fuel. The same compensation was applied to the natural gas and diesel fuel tanks on becoming empty.

In general, for gasoline engines, the ratio of air to fuel-type delivered into the combustion chamber is referred to as the stoichiometric ratio. The stoichiometric ratio for natural gas (methane) is 17.2:1. A perfect diesel fuel stoichiometric ratio is 14.6:1. These are considered lean ratios, and ideally, an engine uses a ratio that is slightly less for maximum torque [12]. The ratio for E-85 ethanol is much lower, 8.47:1, for a lean ratio [13]. For our diesel-based system, the stoichiometric ratio was scaled from 14.6:1 to 8.4687:1 based on the mixture. The algorithm to calculate this took the difference of the two ratios 14.6 − 8.47 = 6.13, multiplied that by the amount of diesel fuel in the ratio, and added that to 8.467. For a 50%-50% scale, respectively for diesel and ethanol, the ratio was (6.23 × 0.5) + 8.47 = 11.585:1.

The ideal time for a spark plug to spark was not determined (by software) in the simulation. The ideal time of sparking the fuel, including the combustion period is normally determined by the O₂ sensor readings for the amount of oxygen and fuel in the
exhaust. The timing function was not simulated for the lack of a reliable API and due to the stochastic behavior and other difficult-to-simulate effects of the engine on timing. Instead, we inferred from the simulation that the spark of the fuel did affect the state of the engine because of the feedback aspect in the cycles of combustion in the ECU/FCM coupling.

3.4 The Engine

The Engine class of the simulator was implemented using the principle of a crankshaft angle. Typically, during each combustion cycle, a crankshaft rotates twice, that is 720 degrees. A throttle provides a scaling unit for the Rotations-Per-Minute (RPM) of the crankshaft between an idle speed and maximum RPM. Crankshaft angles are incremented in real time based on the determined RPM. The crankshaft angle in this simulation was controlled by the RPM, which is in turn controlled by the throttle.

The concept of spark timing using a crankshaft angle that is updated in real time presents a programming problem that simulates a physical engine well. Since the crankshaft angle was updated so frequently, often in the tens of milliseconds, sending the spark signal to the engine at the correct time became difficult. The difficulty normally stems from the stochastic behavior of the engine, the probability distribution of which was not known a priori, as indicated above. To compensate for this issue in the simulation, the system captured the crankshaft at a range of angles between predetermined ranges of angles. This also simulated the concept of spark plug misfire.

4. Data Analysis and Results

Table 1 represents the simulation data that were observed. The Stress values indicate the level of stress on the Engine. Combustion Time represents the time it took to complete one combustion cycle. Throttle Position represents the amount of pressure applied to the accelerator pedal which was simulated by the pressure sensor. When there are values in the Percent of Ethanol and Percent of Diesel columns in the table, the values indicate that the Engine used a mixture of fuel in their respective proportions. When there are no values for the fuel type the Engine used either natural gas or 100% diesel fuel. These values were calculated in the FCM. RPM values were captured in the simulation based on the position of the throttle and were scaled according to a maximum RPM value, which in our case was set to 5,000.

It was observed from the simulation data that as the stress level increases the RPM increases monotonically as well; the converse was also observed. Similarly, as depicted in the corresponding graph in Fig. 5, as the stress increases in response to the RPM increase, the combustion period decreases precipitously as well. These observations support the initial expectations that: As the hybrid car runs on natural gas on a leveled road and comes to an incline, the stress increases. The throttle level was also increased to compensate for the stress, which signaled the ECU to switch to a fuel mixture to increase the output of the Engine.

<table>
<thead>
<tr>
<th>Stress</th>
<th>Combustion period (ms)</th>
<th>Throttle position</th>
<th>% Ethanol</th>
<th>% Diesel</th>
<th>% Natural gas</th>
<th>RPM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>800.0</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>100</td>
<td>150</td>
</tr>
<tr>
<td>2</td>
<td>600.0</td>
<td>4</td>
<td>-</td>
<td>-</td>
<td>100</td>
<td>200</td>
</tr>
<tr>
<td>4</td>
<td>400.0</td>
<td>6</td>
<td>-</td>
<td>-</td>
<td>100</td>
<td>300</td>
</tr>
<tr>
<td>8</td>
<td>240.0</td>
<td>10</td>
<td>-</td>
<td>-</td>
<td>100</td>
<td>500</td>
</tr>
<tr>
<td>7</td>
<td>218.2</td>
<td>11</td>
<td>-</td>
<td>-</td>
<td>100</td>
<td>550</td>
</tr>
<tr>
<td>17</td>
<td>88.9</td>
<td>27</td>
<td>-</td>
<td>-</td>
<td>100</td>
<td>1350</td>
</tr>
<tr>
<td>38</td>
<td>66.7</td>
<td>36</td>
<td>74</td>
<td>26</td>
<td>-</td>
<td>1800</td>
</tr>
<tr>
<td>49</td>
<td>49.0</td>
<td>49</td>
<td>52</td>
<td>48</td>
<td>-</td>
<td>2450</td>
</tr>
<tr>
<td>49</td>
<td>45.3</td>
<td>53</td>
<td>52</td>
<td>48</td>
<td>-</td>
<td>2650</td>
</tr>
<tr>
<td>73</td>
<td>34.3</td>
<td>70</td>
<td>4</td>
<td>96</td>
<td>-</td>
<td>3500</td>
</tr>
<tr>
<td>71</td>
<td>33.3</td>
<td>72</td>
<td>8</td>
<td>92</td>
<td>-</td>
<td>3600</td>
</tr>
<tr>
<td>74</td>
<td>31.6</td>
<td>76</td>
<td>2</td>
<td>98</td>
<td>-</td>
<td>3800</td>
</tr>
<tr>
<td>78</td>
<td>30.4</td>
<td>79</td>
<td>-</td>
<td>100</td>
<td>-</td>
<td>3950</td>
</tr>
<tr>
<td>85</td>
<td>29.6</td>
<td>81</td>
<td>-</td>
<td>100</td>
<td>-</td>
<td>4050</td>
</tr>
<tr>
<td>89</td>
<td>29.3</td>
<td>82</td>
<td>-</td>
<td>100</td>
<td>-</td>
<td>4100</td>
</tr>
<tr>
<td>94</td>
<td>28.9</td>
<td>83</td>
<td>-</td>
<td>100</td>
<td>-</td>
<td>4150</td>
</tr>
<tr>
<td>99</td>
<td>28.6</td>
<td>84</td>
<td>-</td>
<td>100</td>
<td>-</td>
<td>4200</td>
</tr>
<tr>
<td>100</td>
<td>28.2</td>
<td>85</td>
<td>-</td>
<td>100</td>
<td>-</td>
<td>4250</td>
</tr>
</tbody>
</table>
5. Assumptions and Limitations

Under certain ideal situations, it is often very difficult to accurately simulate a complex piece of machinery using software. An engine uses mechanical components to control valves opening and closing, crank position, and fuel injection. Because of this, one of the things not simulated was the car’s O2 sensor. The O2 sensor normally resides in the car’s exhaust and measures the amount of excess fuel or oxygen in the exhaust. If the car has too much oxygen or fuel, the air-fuel ratio is corrected to correct the issue. Our simulation assumed perfect combustion and therefore disregarded the notion of an O2 sensor.

Magnetti Marelli’s team performed air-fuel ratio adjustments and spark timing adjustments based on the O2 sensor reading as well, but using physical components for the hybrid car. Their system tweaked these values to determine the ratio of ethanol to gasoline experimentally. The amount of fuel and air used during each combustion cycle may also pose a problem. For the sake of time and simplicity, an equal amount of air and fuel was used during each cycle. For this reason, torque is not implemented in the simulation, but rather, torque was determined by the quantity of the air and fuel mixture [11].

6. Conclusions

This paper is on the design and simulation of a sensor-based ECU/FCM system for a diesel engine powered hybrid vehicle. The components of the ECU/FCM system were discussed as well as the interactions of the subsystems and the impact on the performance of the system. Furthermore, tilt and pressure sensors were used in the design of the experiment. The sensors were connected to the LEGO NXT unit for interfacing to the simulator.

The software aspect of the research project combined object oriented development with real time embedded systems design and development [14]. Without the use of the Java programming language to represent high level objects the project would have been much more difficult. The ability to combine object-oriented development with embedded systems is becoming more and more prevalent as microcontroller hardware becomes faster, smaller, cheaper, and larger memory capacity. The LEGO NXT platform provided an excellent means to learn and experiment with this type of embedded systems development.

We argued that having different types of fuel in certain proportions, or ratios, of their mixtures would yield gains in performance (gain in the Engine’s output). The performance data were collected through the software simulation, and readings from sensor output were logged into data files. From the analysis of the logged data, it was observed, under the assumptions made, that the proper selection of fuel type influences the performance of the Engine depending on the state of the hybrid car.

References

Embedded Software Control of a Hybrid Vehicle’s ECU/FCM System


