

Control of the Hybrid Engine of the REMUS 600

by

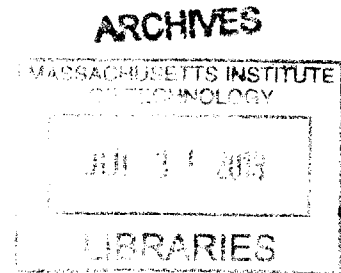
Cindy E. Oh

Submitted to the  
Department of Mechanical Engineering  
in Partial Fulfillment of the Requirements for the Degree of  
Bachelor of Science in Mechanical Engineering

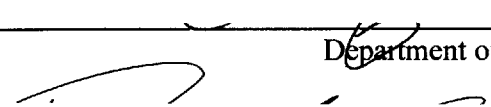
at the

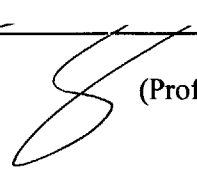
Massachusetts Institute of Technology

June 2013



© 2013 Massachusetts Institute of Technology. All rights reserved.

Signature of Author: \_\_\_\_\_  
  
Department of Mechanical Engineering  
May 13<sup>th</sup>, 2013

Certified by: \_\_\_\_\_  
  
(Douglas Hart)  
(Professor in the Department of Mechanical Engineering)  
Thesis Supervisor

Accepted by: \_\_\_\_\_  
Anette Hosoi  
Professor of Mechanical Engineering  
Undergraduate Officer

# Control of the Hybrid Engine of the REMUS 600

by

Cindy E. Oh

Submitted to the Department of Mechanical Engineering  
on May 13th, 2013 in Partial Fulfillment of the  
Requirements for the Degree of  
Bachelor of Science in Mechanical Engineering

## ABSTRACT

The objective of this thesis was to determine the behavior of the engine of the REMUS 600, an autonomous underwater vehicle. The first step was to evaluate the closing and opening of the choke and throttle valve and write a code that allows the user to easily control the stepper motors that attach to each valve. Then, the engine was to be started, brought up to an ideal speed, and stabilized. All these processes were done through a PIC C compiler and a microcontroller. Testing revealed that it took  $10.3 \pm 0.2$  and  $113.6 \pm 1.7$  steps to open the choke and throttle respectively and  $4.5 \pm 0.1$  and  $206.4 \pm 2.2$  to close the choke and throttle respectively. Code to perform the aforementioned has been preliminarily tested and successful, which should allow future users to maintain control of the engine speed and power output.

Thesis Supervisor: (Douglas Hart)

Title: (Professor in the Department of Mechanical Engineering)

## Acknowledgments

The author would like to thank Professor Doug Hart and Mariah Murray for their extensive help in writing this thesis. It was a pleasure working with Mariah and Cappie, and this work could not have been completed without them. Lastly, the author would like to thank her wonderful family and friends who, although helped her procrastinate, also empowered her to finish. Thank you.

## Table of Contents

1. Introduction.....	5
1.1 History of Underwater Vehicles .....	6
1.2 Autonomous Underwater Vehicle Technology Overview .....	7
1.2.1 Autonomy.....	7
1.2.3 Energy.....	8
1.2.3 Navigation .....	8
1.2.4 Sensor Systems.....	9
1.2.5 Communication .....	10
2. Purpose.....	10
2.1 Vehicle Overview.....	11
2.1.1 Engine System .....	11
2.1.2 Fuel System .....	12
2.1.3 Snorkel System .....	13
2.1.4 Control System .....	13
3. Engine Control.....	13
3.1 Choke & Throttle Control .....	13
3.1.2 Issues Encountered During Testing .....	19
3.2 Starting the Engine .....	19
3.3 Engine Rotational Speed & Control Loop.....	21
4. Results & Conclusions.....	25
References.....	26
Appendix.....	27
A1.1 .....	27
A1.2 .....	29
A1.3 .....	30

[Type text]

## **1. Introduction**

Underwater vehicles have proven to be an important development for navies all around the world. They allow engineers to bypass a multitude of constraints that a team of human divers have in underwater endeavors, allowing an infinite amount of knowledge to be more readily available. Hydrographic reconnaissance, the study of an area of water to determine key characteristics governing the water depths, costal features, beach gradients, and marine life as well as the location of natural ocean plants and manmade obstacles, has grown significantly after the development of efficient underwater vehicles. Additionally, oceanographers have equipped these vehicles with sensors that can measure properties such as the ocean's conductivity, turbulence, pollutants, dissolved oxygen content, and temperature. With a more accurate measurement technique of these parameters, meteorologists can obtain a better understanding about weather events (ie hurricanes), scientists can manage the water quality with more ease, and climate modelers can have a better understanding of changes over time.<sup>3</sup>

The most essential advantage that underwater vehicles provide, and likely the biggest motivation to expand this technology, though, is that expanded access of underwater information gives crucial information to navies and mine countermeasures forces. Underwater vehicles can detect mines via high frequency and high-resolution sonar along with low light level television. After detection, the vehicles are able to dispose of or neutralize mines using cable cutters and explosive charges. Although the development of

[Type text]

underwater vehicles is relatively new, already they are looking to be promising option for ocean-based research and in mine countermeasures because they are able to gather data from water shallower than what a boat can access, as well as water deeper than what a human diver or tethered vehicle can reach.

## **1.1 History of Underwater Vehicles**

Although there is quite a variety in underwater vehicles, they can be categorized into two main classes: manned and unmanned systems. Focusing on unmanned systems, there are furthermore two subsections within this category: remotely operated underwater vehicles and autonomous underwater vehicles. This thesis will focus on the latter. It is informative to understand the development of autonomous underwater vehicles, because it gives insight into why the technology has evolved into what it is today.

Although Robert Whitehead is credited with the first autonomous underwater vehicle (1866), the vast development of these vehicles began in the 1960s. At this stage, most vehicles could only perform very basic and very specific applications, but many were motivated by the need to obtain oceanographic data along with precise trajectories. Within the next 10 years, a number of testbeds were built, and a great amount of funds were allocated in hopes of understanding the potential of these autonomous systems. During the 1980s, experiments were done with proof of concept prototypes. Concurrently, exponentially smaller and more powerful computing power was available which offered the potential of implementing complex guidance and control algorithms on autonomous platforms. As funding for these projects increased, the technology of the operational

[Type text]

systems improved significantly. This decade was especially a turning point in regard to the implementation of a vision system of these autonomous underwater vehicles. A particularly famous system called the 'Advanced Unmanned Search System' (AUSS) carried 20 kilowatt-hours of energy with the use of silver zinc batteries. Moving forward, there were advances made that allowed for more precise control as well as more defined missions. Autonomous underwater vehicles are now widely used in navies and even available commercially. <sup>4</sup>

## **1.2 Autonomous Underwater Vehicle Technology Overview**

Because this field is still relatively new, there are many opportunities and necessary improvements still to be made. There are five main technologies that blanket the system and they include: autonomy, energy, navigation, sensors, and communications. Of course this is by no means an all-inclusive list, but rather a categorization of the most important aspects.

### **1.2.1 Autonomy**

In the 1980s, autonomous underwater vehicles did not have the ability to accomplish the assigned tasks because they did not possess the level of intelligence and control necessary. Mission planning had to be extremely strategic in order to allow a vehicle to understand a string of simple commands and perform a relatively involved task. Additionally, the vehicle had to have sufficient perception and situation assessment abilities so that it could respond accordingly during a mission. This presented a unique challenge because engineers were not able to preprogram a set list of commands with absolutely certainty of what the machine would encounter. There was considerable effort but diminished results in regard to intelligent systems architecture design. To date, the challenge autonomy presents

[Type text]

is still overwhelming, and although there have certainly been advances in autonomy, these have not been integrated into autonomous underwater vehicle technology.

### **1.2.3 Energy**

Since the value of autonomous underwater vehicles come from the ability to remain untethered, the energy system and energy management is extremely important and often a limiting factor. In the early 1970s, only a few hours of underwater life was achieved by the vast majority of autonomous vehicles. Over the course of the last few decades, some systems were made to be capable of missions on the order of magnitude of days. Most can achieve the life span of 10s of hours, and a very few can endure years underwater. In earlier years, most systems used Lead Acid batteries. Some used Silver Zinc batteries, but this was almost an order of magnitude more expensive. Lead Acid evolved into Lithium primary batteries, but in the late 1980s, advances in Nickel-metal hydride batteries presented a promising direction. Battery power is to date the most/ only feasible option for autonomous underwater vehicles, but solar and heat energy technologies are also being taken advantage of to assist in increasing the life span of each vehicle.

### **1.2.3 Navigation**

Navigation is undoubtedly crucial for being able to autonomously complete missions. The earlier systems used dead reckoning for their navigation, meaning that the vehicle would calculate its current position by using a previously determined position. Not only did this method accumulate errors very quickly, it also needed precise input for speed and direction. Acoustic transponder navigation systems and inertial navigation systems were an



[Type text]

improvement in performance but were significantly more costly. Recently, Global Positioning Systems (GPS) has been integrated with many autonomous underwater vehicles, allowing it to be possible to obtain accurate positioning data when the vehicle surfaces. Navigation is the area that has made the most improvement since the early autonomous systems.

#### **1.2.4 Sensor Systems**

Sensors necessary for carrying out fundamental functions of an autonomous underwater vehicle fall into the other categories of autonomy, energy management, and navigation. In this subsection, only the sensors used to take data are addressed. Most efforts have been made to incorporate already existing sensors into an autonomous underwater platform, but basic sensor technologies have by far preceded the reliable vehicle operation technologies. With the evolution of autonomous systems, it became blatantly clear that underwater autonomy imposed large constraints and that entirely new sensors needed to be developed. Improving the sensor's ability to intake/output more data, to consume less power, to be more reliable, and to be smaller in size would prove exponentially advantageous to overall machine performance.

Additionally, vision systems are continuously being made with higher and higher resolution imaging, both optically and acoustically. The technology to acquire high resolution data at a distance exists, but the perception technology does not. The limiting factor is the vehicle's lack of ability to autonomously process this data and utilize it for real-time guidance and control decisions.

[Type text]

### **1.2.5 Communication**

Laser communication (at short ranges) and radio frequency current field communications (at long ranges) are available and utilized, but by far the most widespread system is acoustic communications. In the past few decades, there has been exponential growth in acoustic communication which has allowed it to have a range on the order of kilometers and relatively low error rates. Acoustics relies on hydrophones which are designed to record, listen to, and analyze, underwater sound via a piezoelectric transducer. It's crucial for communication systems to be standardized and for the design to be guided by the mission tasked to the vehicle. <sup>2</sup>

## **2. Purpose**

The autonomous underwater vehicle that will be the focus of this thesis is the REMUS 600. There were four major considerations in the design of this project. The first was an energy system that would allow the autonomous system to engage in missions for 40 consecutive 12 hour missions before a necessary surfacing to recharge. Although as mentioned in section 1.2.3, the energy systems for autonomous underwater vehicles have drastically improved, most vehicles are still extremely limited by the lack of storable energy. The second consideration was safety. High power density proves useful in such systems, but since there will be no inspection during mission, it was important to have a high factor of safety. The third consideration was feasibility. The end goal was to produce a working prototype, so newborn technologies that would require extensive funding and time were not considered. While there exist future recommendations with technologies that may be

[Type text]

superior to what was chosen for the REMUS 600, it was prioritized as a short term design project. Lastly, the fourth consideration was size. The REMUS 600 is less than thirteen inches in diameter, so the design called for a compact and dense architecture.

## **2.1 Vehicle Overview**

The engine supplies the power for the REMUS 600 to perform its missions. In order to stably and reliably output power, it needs the correct composition of air and fuel. The subsequent subsystems contribute to this regulation of fuel composition or the stabilizing of the vehicle itself.

### **2.1.1 Engine System**

The engine used in the REMUS 600 is a Honda GXH50 because of its availability, size, and power output. This engine was taken from a Honda EU1000i generator, but then modified in order to suit this autonomous underwater application. The generator was modified with hand-wound coils as shown below in Figure 1. With this, it was possible to change the crank start motor to an autonomous brushless motor. The generator produces about 1 kiloWatt of power at about 16% efficiency. Within the engine system, there is a modified fluid heat exchanger. In its original application, the Honda engine is cooled by forced convection, but as the engine is mounted internally, a water heat exchanger had to be integrated. Lastly, a Hall effects sensor was incorporated in order to acquire rotational speed data.

[Type text]

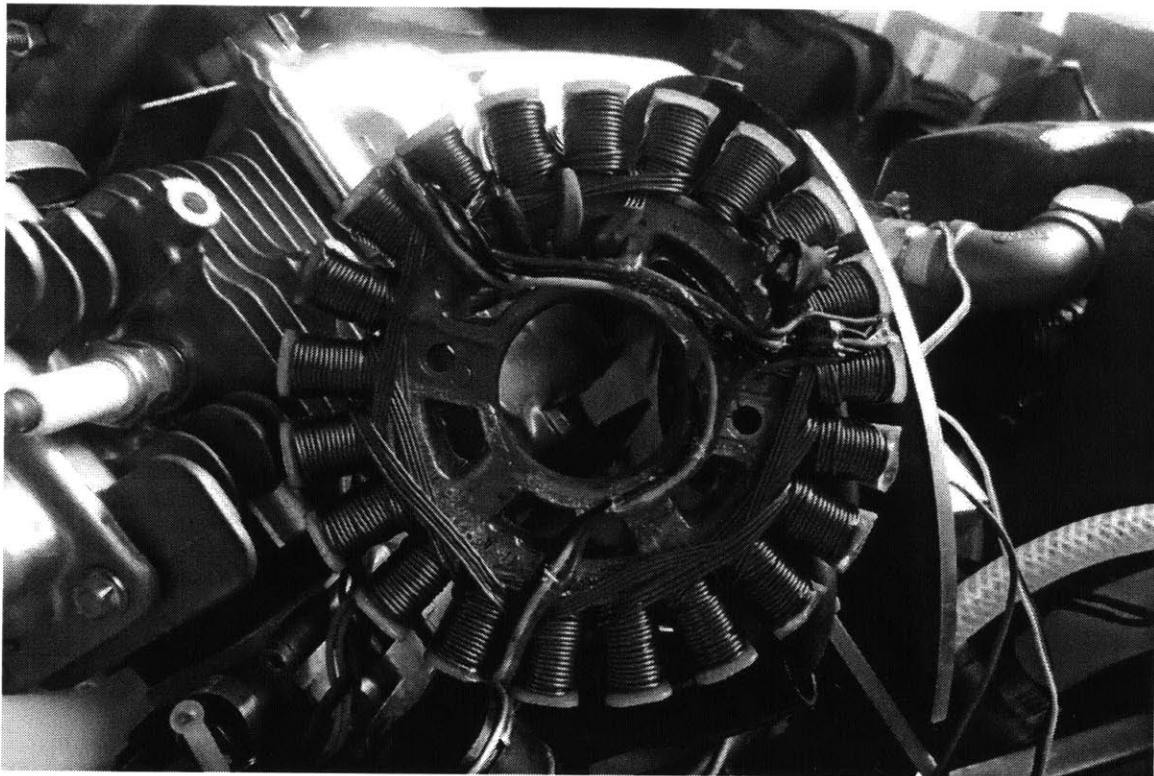


Figure 1: Customized, hand wound generator. (Modified from a GXH50)

### 2.1.2 Fuel System

The fuel is stored in bladders found in a sealed tank at either end of the REMUS 600. This fuel is supplied to the engine via the passive regulation of piping, pumps and valves. However, as the fuel is consumed by the engine, the reduction of weight (of the entire vehicle) presents a problem. In order for the vehicle to maintain a zero moment and as well as its original weight (which will maintain neutral buoyancy), the REMUS 600 must intake water and air to replace the consumed mass of the fuel. The fuel system delivers fuel to the engine as it fills another bladder with water. The design used in the REMUS 600 minimizes the opening in the hull.

[Type text]

### **2.1.3 Snorkel System**

The snorkel system assists the fuel system in maintain neutral buoyancy by intaking air. This system also intakes water and routes it to the cooling system (the aforementioned heat exchanger) to maintain proper engine temperatures. With this system, the REMUS 600 is able to operate in harsh rough seas with integrated redundancies for safety reasons and a series of valves and baffles which remove water from the airflow.

### **2.1.4 Control System**

This is the final overarching governing system which manages the power generation of the REMUS 600. It controls the entire process of recharging. A control board interfaces with the REMUS computer, and it runs commands and manages system errors. An engine mounted starter-generator generates power, which is then filtered into a form that can charge the batteries and supply power to other systems.

## **3. Engine Control**

### **3.1 Choke & Throttle Control**

While all previous systems are involved with the engine, there exists a more internal method of control within the engine system. The direct control of the engine fuel intake is done by the choke and the throttle. The choke valve allows or prevents intake of air and the throttle valve allows or prevents the intake of gasoline. Figure 2 below shows the engine and the location of the choke and throttle.

[Type text]

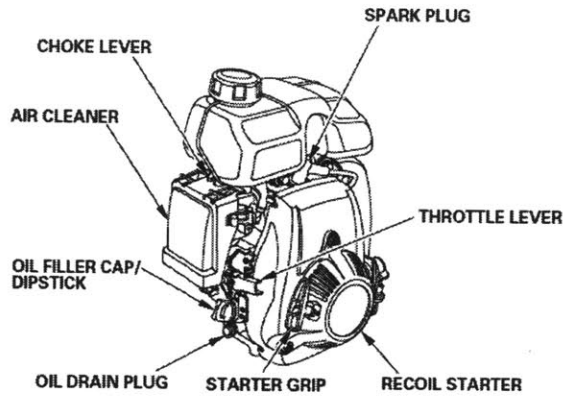


Figure 2: Diagram of the original Honda GXH50 motor used. In the REMUS 600, the fuel tank is removed and stored separately. The choke lever and throttle level are control with two stepper motors that are fixed to one mount. (Ref: Honda Manual)

In the REMUS 600 engine system, the stepper motor that controls the opening and closing of the choke valve was modified. Figure 3 below shows both the choke and throttle lever arms. These connect to two stepper motors shown in section 3.1.2 in Figure 6.

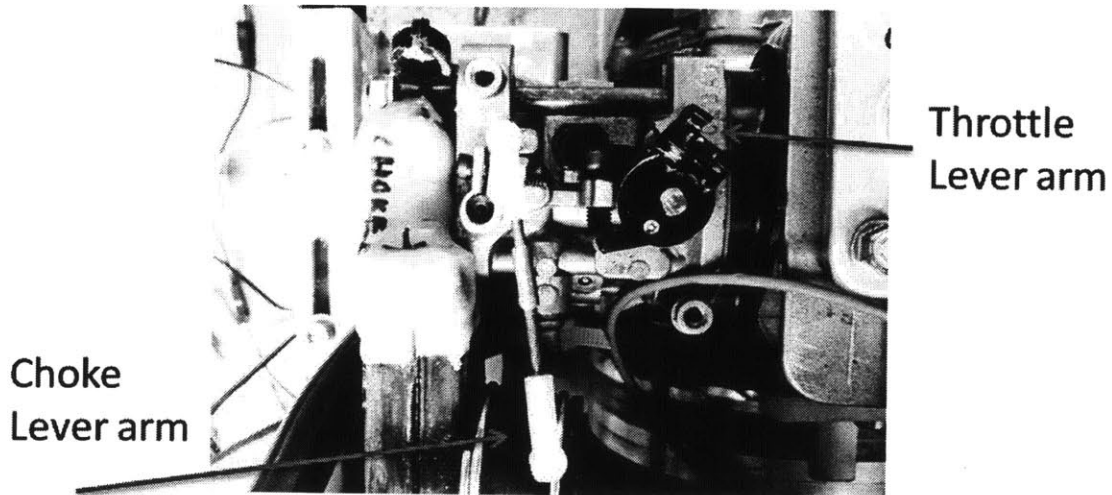


Figure 3: Top view of the engine showing the choke and throttle valves and lever arms. These lever arms connect to the stepper motors shown in Figure 6.

[Type text]

Before any control of the engine could be performed, testing had to be done in order to know precisely how many steps it takes for the stepper motors to completely open and completely close each respective valve. The results for the choke and throttle are shown below in Table 1, with 95% confidence.

Table 1: Results (with 95% confidence) of the number of steps it takes for the choke and throttle valves to completely open or close.

	Choke	Throttle
Open	10.3 ± 0.2	113.6 ± 1.7
Close	4.5 ± 0.1	206.4 ± 2.2

The setup and a block diagram are shown beneath in Figure 4. As demonstrated the microcontroller is connected to each stepper motor driver via two pins: the step line and direction line. Each will be explained at the end of this section as the code controlling this setup is explained.

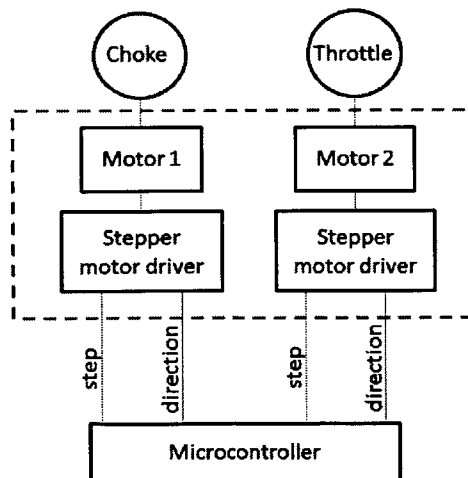


Figure 4: Block diagram showing the setup to test the stepper motors. The user communicates with the microcontroller which then sends commands to the stepper motor board (outline in dashed box). This board ultimately communicates with the choke and throttle.

[Type text]

The code which controls the stepper motor is given in full in the appendix A1.1. The first 3 lines call header files and define the microcontroller. With these included, the code can identify other commands and can understand the functions it must carry out. Lines 7 through 11 define the pins on the board. Both the throttle valve and the choke valve have two pins to control it. The pins beginning with 'stepDir' tell the valve which direction to move and the pins beginning with 'stepper' tell the valve when to move.

The point of the code is to provide the user with an easy way to control both the choke and throttle valves and subsequently, control the speed of the engine. It recognizes a keyboard input and with one keystroke will open or close either the throttle or choke. Because the throttle takes hundreds of steps to close or open, the code is designed to move the step the throttle multiple times with one keystroke. The variable 'NumOfSteps' can be inputted before running the code in order to control how many steps the throttle will move in response to one keyboard event. The tests were run with 'NumOfSteps' equal to eight.

When there is a signal sent to a 'stepDir' pin, the microcontroller will know which direction the stepper motor should move, but will not move it until the 'stepper' pin receives a signal. In the code 'output\_high' sends the signal, and conversely 'output\_low' stops it. The proper signal for the stepper pin is a square wave. Therefore, the code calls for 'output\_high' and then 'output\_low' with a 15 millisecond delay after each.



[Type text]

There are also 4 variables defined apart from the key strokes. These are in lines 27 to 30 and allow the user to easily keep track of how many steps have been taken. These are: 't\_close,' 't\_open,' 'c\_close,' and 'c\_open,' and they count the number of steps the throttle or choke have been opened or closed.

The table 2 below shows which keyboard inputs are important in this code.

Table 2: Shows 5 key strokes that control the microcontroller. 'y' and 'r' control the throttle valve, and 'v' and 'x' control the choke.

Keyboard Input	Action
Y	Opens throttle
R	Closes throttle
V	Opens choke
X	Closes choke
F	Clears values

Every time a 'y' 'r' 'v' or 'x' is pressed, an if-loop is initiated and runs the correct set of commands. The if-loop acts as a keystroke detector as the user only communicates to the microprocessor if an appropriate key is struck. With this if-loop, 'y' or 'r' will increase 't\_open' or 't\_close' (respectively) by 'Numof Steps.' Alternatively, 'v' or 'x' will increase 'c\_open' or 'c\_close' by one step. The key 'f' will clear all set all four variables to zero so that the user may reset the count. A simplified flow chart of this code is shown in Figure 5.

[Type text]

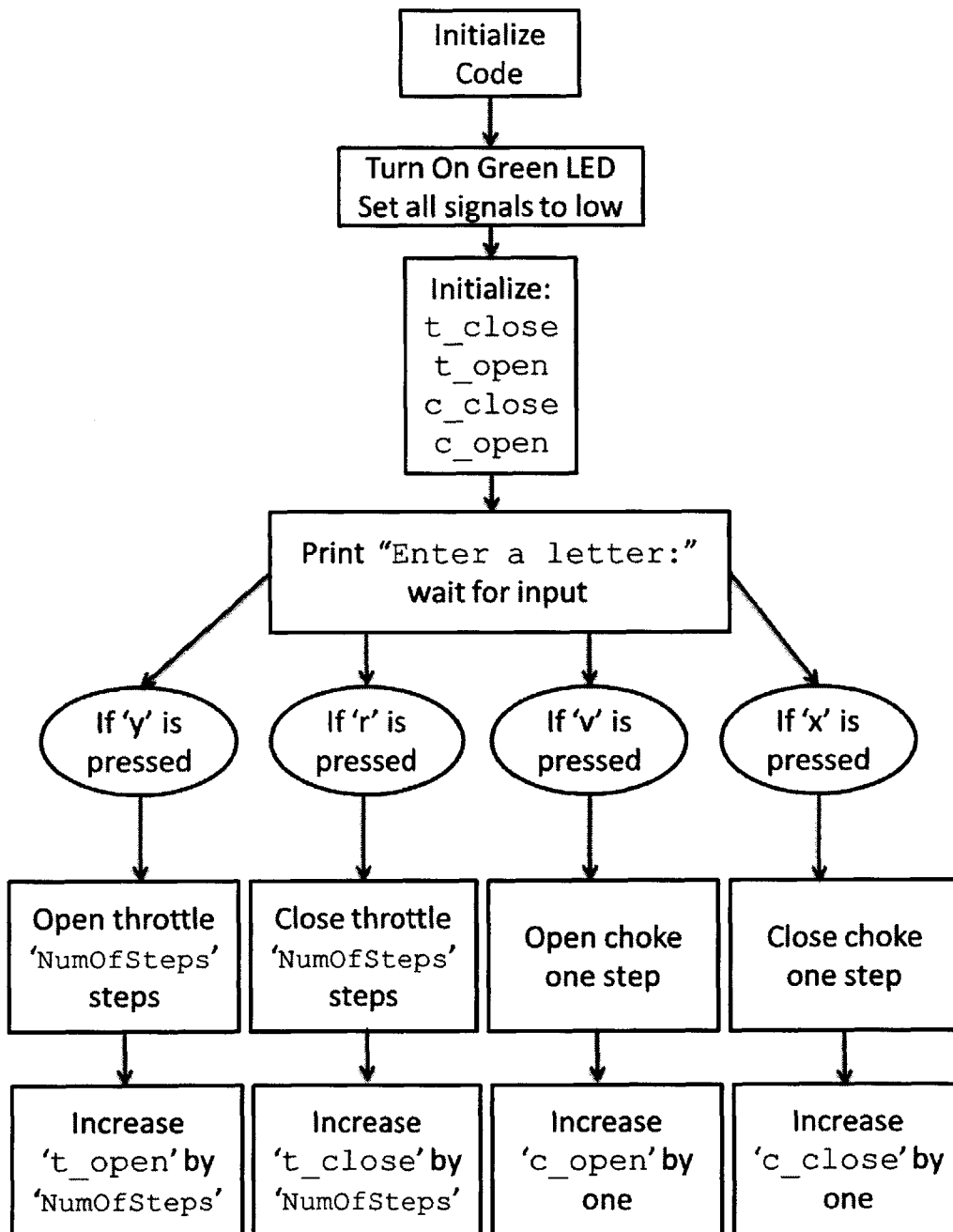


Figure 5: Flow chart of the code. After initialization, the code waits for user input and sends according signals to the microcontroller.

[Type text]

### 3.1.2 Issues Encountered During Testing

#### *Excess forces on the Lever Arms*

After unreliable results of testing the throttle, it was evident that the stepper motor would stall for an unpredictable amount of steps at a time. The best possible explanation was that the orientation of the stepper motors put excess loads on the valve levers that exceed the output force of the stepper motor. By drilling and tapping more holes and allowing for different orientations, experiments were run to optimize the mounting position of the motors. Figure 6 below shows the revised mount with the extra allowable positions.

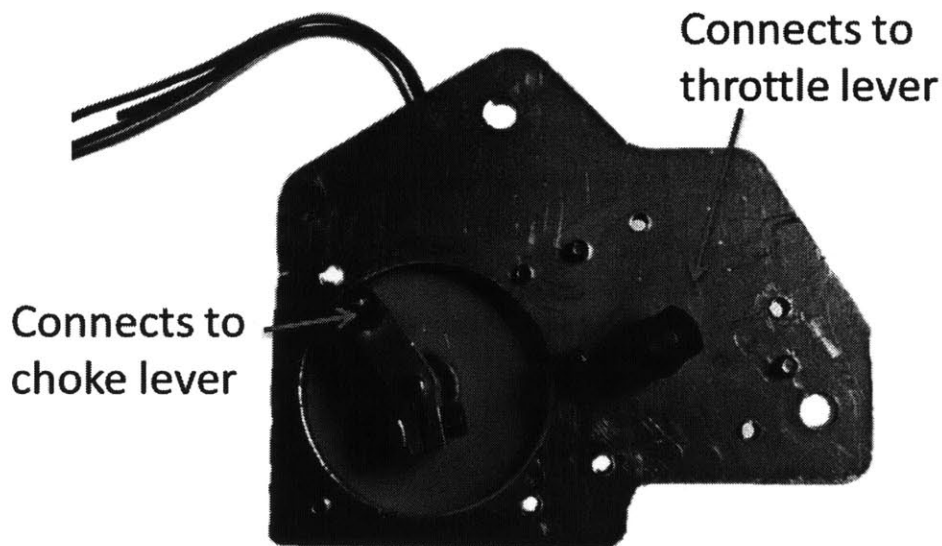


Figure 6: Mount of the stepper motor controlling the opening and closing of the choke and throttle valves. Additional drilled and tapped holes allow the user to change motor orientation.

### 3.2 Starting the Engine

As mentioned in the engine description, the engine was modified to be a brushless autonomous start engine. As the generator starts the engine, the engine must draw fuel via

[Type text]

the throttle and begin its cycle so that the engine can run independently from the generator.

The code that allows the user to communicate to the microcontroller and the generator is given in the appendix A1.2. The preamble is identical to the other codes. When the code initiates, it prompts the user to press 's' to start the engine. Once the input is received, a single square wave is sent to the proper pin. The variable 'delay' determines the pulse width. Through extensive testing, it has been determined that the optimum value of 'delay' is 1000 or 1 seconds. (1000 microseconds). With a variance of fuel-air ratios, it is recommended that the user tries different values of this delay, to obtain a reliable engine start. Figure 7 below shows a block diagram of the set-up used.

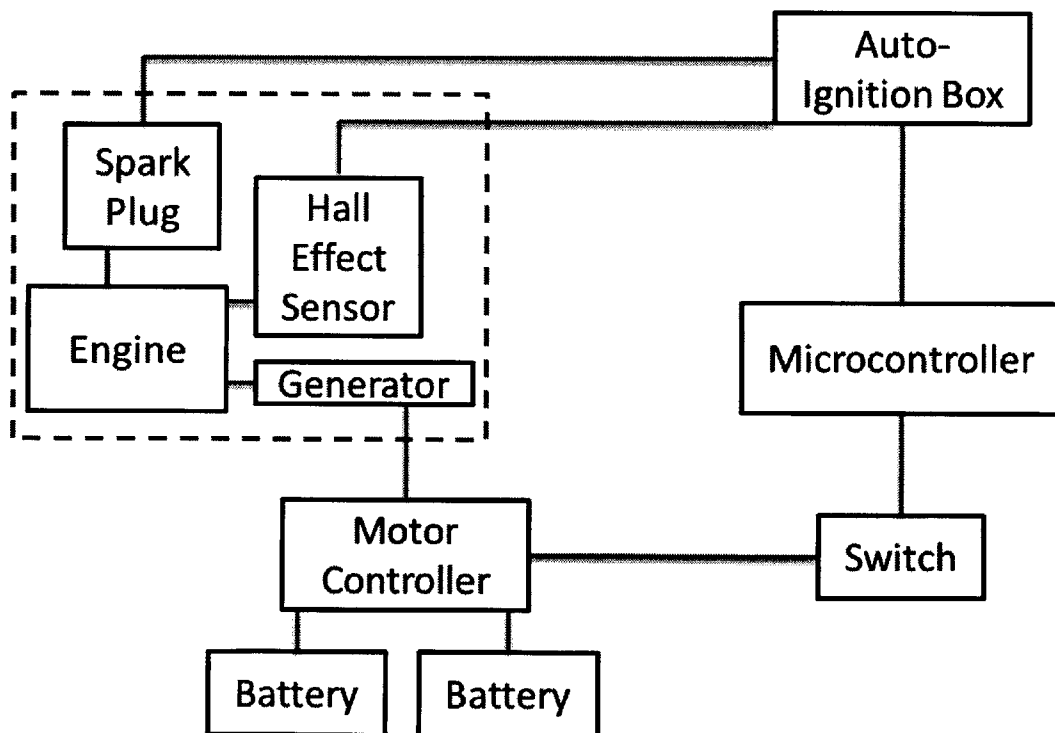


Figure 7: Block diagram showing the test setup used to start and time the engine.

[Type text]

### 3.3 Engine Rotational Speed & Control Loop

Once there is full control of the choke and throttle valves, the next step is to acquire data about the rotational speed of the motor. As mention in section 2.1.1, a Hall effect sensor was integrated into the engine system. A Hall effect sensor is a transducer that responds to a change in a magnetic field by output a variable voltage. There is a magnet that rotates with the engine in the system, and every time that magnet passes the Hall effect sensor, the sensor outputs a signal to a pin. The magnet and Hall effect sensor can be seen in the circled part of figure 8.

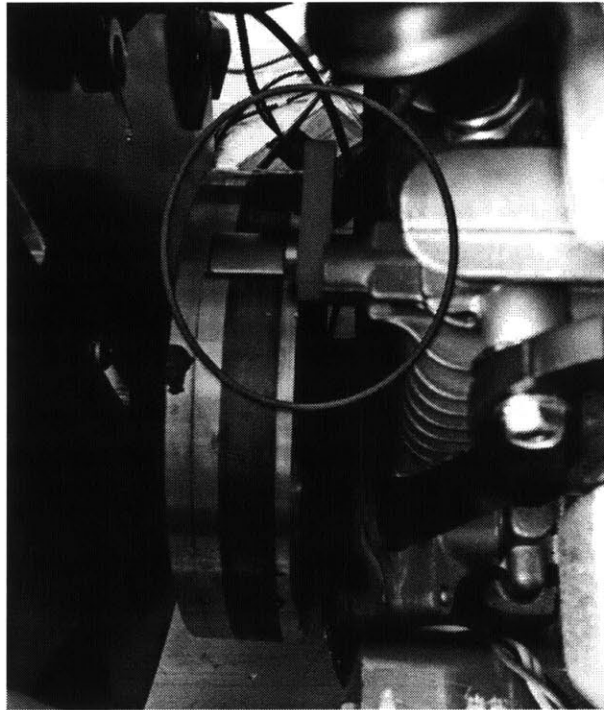


Figure 8: Highlighted in this figure is the magnet and Hall effect sensor. As the magnet passes the sensor, a signal is sent to the microcontroller.

The code that acquires rotational speed data is included in the appendix A1.3. As the magnet passes the Hall effect sensor, a signal is generated in the form of a peak over a finite amount

[Type text]

of time. Figure 9 below is a simplified version of the general shape of the signal.  $Time_1$  is the duration of time that the magnet is in close enough proximity of the Hall effect sensor, and  $Time_2$  is the duration of time in which the rest of cycle is happening. Thus, the duration of one cycle is the sum of both times and will be referred to as cycle time.

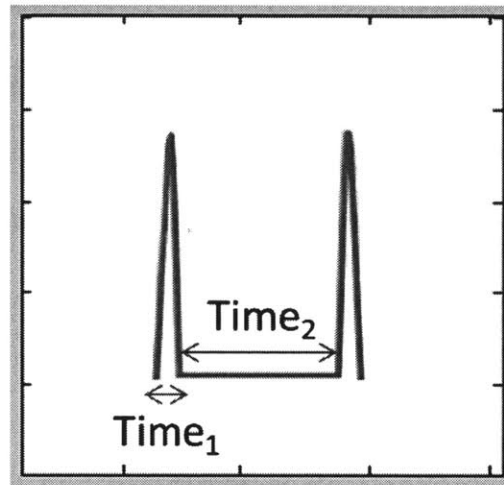


Figure 9: Typical shape of signal due to the magnet passing the Hall effect sensor. The peak last for a finite amount of time that must be measured.

The code shown in the appendix is used to obtain this cycle time. Line 1 – 10 is an identical preamble which is explained in section 3.1. By using an internal timer called 'setup\_timer1' the user is able to time the length of any event. The interrupts that are enabled and disabled in lines 28, 31, 36 and 44 are set in place in order to account for the possibility of the timer running out of memory. The code sets the variable 'overflow\_count' to zero before the initialization of the timer (this is essentially the memory limit). As the timer's memory fills up, the magnitude of the variable 'overflow\_count' also increases. Lines 38 and 46 of the code allow the user to keep

[Type text]

track of the time even if the timer memory overflows and resets. (Line 38 does so for time<sub>1</sub> and line 46 does so for time<sub>2</sub>.) Table 3 below more thoroughly explains the syntax of the code.

Table 3: Explanation of fundamental syntax include in the code (Appendix A1.3)

Syntax	Explanation
unsigned int 16	unsigned means that the variable can only take on (+) values; 16 bit memory
while(input(magnet))	code runs only as long as there is a signal from the magnet passing the Hall effect sensor
while(!input(magnet))	code runs only as long as there is no signal from the magnet passing the Hall effect sensor
get_timer1()	Reads the time off the timer

The first portion of the code (lines 51 and previous) obtains the time it takes for one engine revolution to occur. The second portion of the code (everything thereafter) stabilizes the engine speed. The user sets an ideal time by setting the variable 'ideal' to the desired cycle time in line 16. Then, if the engine is running too fast, ie. the averaged cycle time of 10 cycles is smaller than the ideal cycle time, the throttle will open four steps to intake more fuel and increase the speed. Conversely, if the engine is running too quickly, the throttle valve will close four steps, and slow the engine down. A simplified flow chart of this code is shown below in Figure 10.

[Type text]

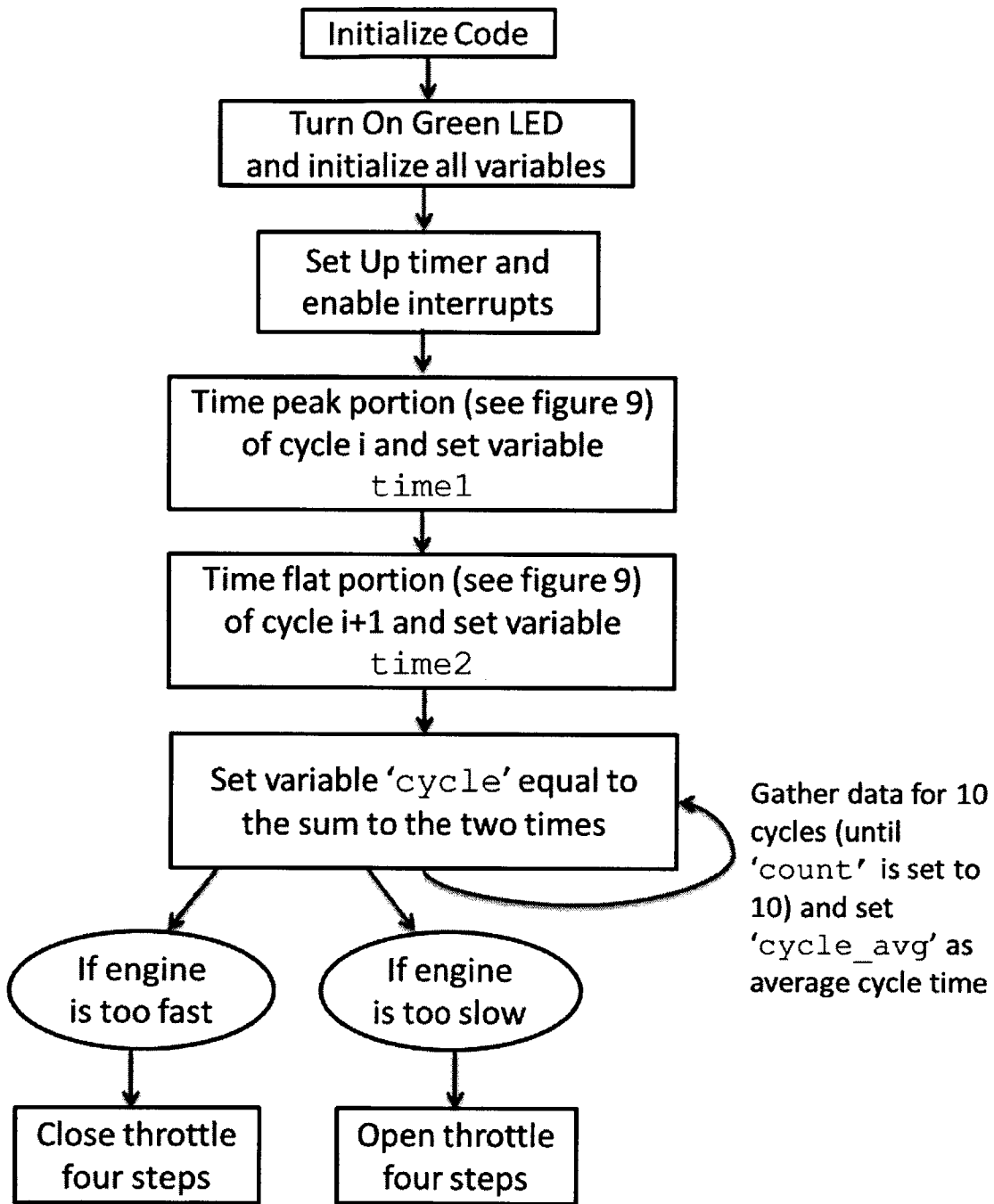


Figure 10: Flow chart of code used to control and stabilize engine speed. Ten cycles are averaged and then compared to the ideal cycle time.



[Type text]

## 4. Results & Conclusions

The engine and engine control of the REMUS 600 are crucial because they regulate the power allocation through the duration of the mission. The combination of written and tested codes allows the user to control the throttle and choke valve, and consequently the engine speed. Testing revealed that it took  $10.3 \pm 0.2$  and  $113.6 \pm 1.7$  steps to open the choke and throttle respectively and  $4.5 \pm 0.1$  and  $206.4 \pm 2.2$  to close the choke and throttle respectively. After creating an internal timer to measure the cycle speed via the Hall effect sensor, it was possible to write a simple control loop for the engine by opening or closing the throttle appropriately. Lastly, by rewiring the engine switch to the microcontroller, code was written such that the user can start the engine with a keystroke.

Further tests should be run to calculate the ideal speed of the engine. This may depend on the mission at hand; for this reason, the written code was made such that the user could easily define the stable engine speed. Future recommendations include a control loop that allows steady increase or decrease of the engine rather than a set ideal speed. In addition, it is worth noting that the code is currently set so that the speed of 10 cycles is averaged and then compared to the ideal speed. The trade-off between increasing and decreasing this number is that a greater number allows for more reliable data, but a lower number allows for a quicker feedback loop. In the future, it'd be advantageous to run tests with a various number of averaged cycles, and thus optimize stability.

[Type text]

## References

2.013 Engineering Systems Design, MIT Rapid Development Group. "Internal Combustion Engine Hybrid Recharging System." *Critical Design Review* (2012): Web. 14 May 2013.

Blidberg, Richard. "The Development of Autonomous Underwater Vehicles (AUV); A Brief Summary." *ICRA* (200): 12. Print.

Crimmins, Denise, and Justin Manley. "NOAA Ocean Explorer: AUVfest 2008: What Are AUVs, and Why Are They Used?." *NOAA Ocean Explorer Home*. N.p., 25 Aug. 2010. Web. 6 May 2013. <<http://oceanexplorer.noaa.gov/explorations/08auvfest/background/aufs/aufs.html>>.

von Alt, Christopher. "Autonomous Underwater Vehicles." *Autonomous Underwater Lagrangian Platforms and Sensors Workshop 1* (2003): n. pag. *Geo-Prose*. Web. 6 May 2013.

[Type text]

## Appendix

### A1.1

```
1 #include <24HJ128GP306.h>
2 #include <prototype.h>
3 #include <math.h>
4 #fuses HS, NOWDT, PR
5 #use delay (clock=20000000)
6
7 #define stepper_throttle PIN_G0
8 #define stepper_choke PIN_G2
9 #define stepDir_throttle PIN_G1
10 #define stepDir_choke PIN_G3
11 #define GREEN_LED PIN_B4
12
13 void main(void){
14     #use STANDARD_IO (D) //set up keyboard input
15     #use STANDARD_IO (G)
16     #use STANDARD_IO (F)
17
18     char start;
19     int i;
20
21     output_high(PIN_B4); //turns on green LED
22     output_low(PIN_G0); //start with all pins no signal
23     output_low(PIN_G1);
24     output_low(PIN_G2);
25     output_low(PIN_G3);
26
27     int t_close=0;
28     int t_open=0;
29     int c_close=0;
30     int c_open=0;
31     int NumOfSteps=8;
32
33     while(TRUE){
34         printf("\r\n Enter a letter: ");
35         start=getc();
36
37         if (start=='r' ){
38             printf("\r\n Throttle Close 8 Steps");
39             output_high(stepDir_throttle);
40             t_close +=NumOfSteps;
41             for(i=0; i<NumOfSteps;i++){
42                 output_high(stepper_throttle);
43                 delay_ms(15);
44                 output_low(stepper_throttle);
45                 delay_ms(15);
46             }
47         }
```

[Type text]

```
48
49 □ if (start=='y' ){
50     printf("\r\n Throttle Open 8 Steps");
51     output_low(stepDir_throttle);
52     t_open +=NumOfSteps;
53 □     for(i=0; i<NumOfSteps;i++){
54         output_high(stepper_throttle);
55         delay_ms(15);
56         output_low(stepper_throttle);
57         delay_ms(15);
58     }
59 }
60
61 □ if (start=='x' ){
62     printf("\r\n Choke Close 1 Step");
63     output_high(stepDir_choke);
64     c_close +=1;
65 □     for(i=0; i<1; i++){
66         output_high(stepper_choke);
67         delay_ms(25);
68         output_low(stepper_choke);
69         delay_ms(25);
70     }
71 }
72
73 □ if (start=='v' ){
74     printf("\r\n Choke Open 1 Step");
75     output_low(stepDir_choke);
76     c_open +=1;
77 □     for(i=0; i<1;i++){
78         output_high(stepper_choke);
79         delay_ms(20);
80         output_low(stepper_choke);
81         delay_ms(20);
82     }
83 }
84
85 □ if(start=='f'){
86     t_open=0;
87     t_close=0;
88     c_open=0;
89     c_close=0;
90 }
91
92 printf("\r\n t_open %i" ,t_open);
93 printf("\r\n t_close %i", t_close);
94 printf("\r\n c_open %i", c_open);
95 printf("\r\n c_close %i", c_close);
96
97 }
98 }
```

[Type text]

## A1.2

```
1 #include <24HJ128GP306.h>
2 #include <prototype.h>
3 #include <math.h>
4 #fuses HS,NOWDT,PR
5 #use delay (clock=20000000)
6 // #use rs232 (baud=9600, UART1, stream=PORT1)
7
8
9 #define GREEN_LED PIN_B4
10 #define engine_switch PIN_G7
11
12 void main(void) {
13     #use STANDARD_IO (D)
14     #use STANDARD_IO (G)
15     #use STANDARD_IO (F)
16     output_high(PIN_B4);
17
18     char start;
19     int delay=1000;
20     while(TRUE) {
21         printf("\r\n Press s to start");
22         start=getc();
23         if(start=='s') {
24             printf("\r\n Engine has been started!");
25             output_high(engine_switch);
26             delay_ms(delay);
27             output_low(engine_switch);
28         }
29     }
30 }
```

[Type text]

## A1.3

```
1 #include <24HJ128GP306.h>
2 #include <prototype.h>
3 #fuses HS,NOWDT,PR
4 #include <math.h>
5 #use delay (clock=20000000)
6 #define stepper_throttle PIN_G0
7 #define stepper_choke PIN_G2
8 #define stepDir_throttle PIN_G1
9 #define stepDir_choke PIN_G3
10 #define magnet PIN_G6
11
12
13 unsigned int16 overflow_count;
14 int i;
15 int cycle;
16 int ideal=1000;
17
18 #int_timer1
19 void timer1_isr(){
20     overflow_count++;
21 }
22
23 void main(){
24     unsigned int32 time1;
25     unsigned int32 time2;
26
27     setup_timer1(TMR_INTERNAL | TMR_DIV_BY_1);
28     enable_interrupts(int_timer1);
29
30     while(TRUE){
31         enable_interrupts(INTR_GLOBAL);
32         while(input(magnet));
33         overflow_count=0;
34         set_timer1(0);
35         while(!input(magnet));
36         disable_interrupts(INTR_GLOBAL);
37         time1=get_timer1();
38         time1=time1+((unsigned int32)overflow_count<<16);
39         time1-=15;
40         while(!input(magnet));
41         overflow_count=0;
42         set_timer1(0);
43         while(input(magnet));
44         disable_interrupts(INTR_GLOBAL);
45         time2=get_timer1();
46         time2=time2+((unsigned int32)overflow_count<<16);
47         time2-=15;
48         cycle=time1+time2;
49         int count=0;
50         count++;
51         printf("Speed is %lu.%06lu rpm. \r\n", (600000000/(time1+time2)),
52             (600/(time1+time2)) %1000000);
53     }
54     while(count <10){
55         int cycle_avg;
56         cycle_avg=(cycle+cycle)/10;
```

[Type text]

```
56
57 if(count=10){
58 break;
59 }
60 if(cycle_avg < ideal){
61 for(i=0; i<4;i++){
62 output_high(stepper_throttle);
63 delay_ms(15);
64 output_low(stepper_throttle);
65 delay_ms(15);
66 }
67 }
68
69 if(cycle_avg > ideal){
70 for(i=0; i<4;i++){
71 output_high(stepper_throttle);
72 delay_ms(15);
73 output_low(stepper_throttle);
74 delay_ms(15);
75 }
76 }
77 }
78 }
79 }
```