Unmanned Ground Vehicle System
Identification and Control

Seung Min Lee, Wei Phing Ang, Stephanie Sumcad, and Charles Tytler†
Purdue University, Department of Aeronautics and Astronautics

This paper provides a frequency response method of system identification for an Unmanned Ground Vehicle. Before system identification could begin, analysis and design of certain hardware components were performed. The accelerometer feedback had to be modified for accurate GPS data. Sensor calibration and installation to the vehicle’s subsystems was necessary as well prior to response analysis. The design of the experiments and analysis of data collected, primarily for the steering subsystem, are discussed, and their involvement in modeling the system in a mathematical form is explained in detail. Also included is the application of controllers to reduce error from input commands for future functionality as an autonomous vehicle.

Introduction

As unmanned aerial vehicles (UAVs) become ever more prominent in the aeronautical industry, more advanced methods of guidance and control must be designed. Although there are many more complications to an aerial vehicle, a simple test bed to apply new navigation and control concepts is the unmanned ground vehicle (UGV). The ground vehicle provides a simplified model for experimentation, by reducing the degrees of freedom from three to two. This, in turn, shortens the testing stages of new control systems and leads to more efficient development before applying them to an airborne system.

In this paper, frequency-domain system identification is applied to a Traxxas Stampede Remote-controlled car to be converted to an unmanned ground vehicle. Specifically, the system identification will be focused on the steering servo and speed controller driving the vehicle. Research was done on the steering and speed dynamics of the vehicle in order to better model the system. Also, a trade on different methods for analyzing the vehicle response was performed to ensure the most accurate results.

MEMSIC 2125 Dual-axis Accelerometer

The MEMSIC 2125 is a dual axis accelerometer sensing system with signal processing. The MEMSIC 2125 can measure acceleration with a range from 0 to ±3 g on either axis. The dynamic and static acceleration is measured with MEMSIC 2125 dual axis accelerometer. Besides, the MEMSIC 2125 dual axis accelerometer can be applied on auto piloting, rotational sensing and tilt sensing. In this project, this MEMSIC 2125 dual axis accelerometer is used to measure the change in position on both sidewise and forward axes.

There are 8 pins on the MEMSIC 2125 dual axis accelerometer. In this research particularly, pin 2, 3, 5, and 8 of the accelerometer were used. Pin 1 is connected to the output of the temperature sensor. Pin 2 gives the digital output of the y-axis acceleration sensor. This is programmable at 100Hz or 400Hz, 100Hz is used in this research. Pin 3 is connected to ground and pin 4 is the power supply for the analog amplifiers in the accelerometer. In addition, pin 5 is the digital output of the x-axis acceleration sensor works at 100Hz. Pin 6, a reference voltage, typically is set to 2.5V. Pin 8 is about the supply input for the digital circuits.

The gravity force was measured in accelerometer as an input to determine the inclined angle. The MEMSIC 2125 dual axis accelerometer can recognize the changes in tilted position if the axis of accelerometer is perpendicular or parallel to the gravity force.

In order to understand the output of this dual axis accelerometer, it should be noticed that the MEMSIC 2125 has 2 duty cycle outputs (x,y). Typically, the total length of total cycle of one period is programmed originally on 2.5 ms or 10ms. However, 10ms was used as the total length in this project. The duty cycle was calculated as the length of the “on” portion of the cycle over the length of the total cycle. In other words, the “on” portion refers to the reaction time in each cycle when the accelerometer detects changes in its positions relative to the gravity. By using fixed sensitivity factor as the value of 12.5% duty cycle per gravity and zero gravity output as the value of 50% of duty cycle, the acceleration for each axis can be calculated by the equation shown in Figure2. By that, we can monitor the change in position and the speed of the vehicle.

![Figure1: Top view of MEMSIC 2125.](image)

![Figure2: Typical output duty cycle.](image)
HMC1512 Magnetic sensor

Initial considerations for the vehicle sensors included the Hall-sensor, potentiometer, and magnetic sensor. While sensors utilizing the Hall-effect would be ideal for measuring wheel revolutions for analyzing the speed controller, they would have little benefit for other subsystems. The potentiometer was considered for steering angle analysis, however all models found experienced high damping and would significantly reduce the accuracy of their readings. The magnetic sensor, specifically the Honeywell HMC1512, provided the best solution as it could sense rotation and linear displacement while avoiding any physical contact with the system.

The HMC1512 sensor was later used instead of the dual-axis accelerometer because it is capable of measuring the direction of the steering wheels and determining the speed of the vehicle as well. Basically, HMC is a magnetic sensor that detects magnetic flux and converts it into voltage that can then be read using the Arduino board or an oscilloscope. The idea of this method was simple. Two very strong magnets were mounted on the motor axis that connects the steering wheels. The sensor is then attached above the magnets. When the vehicle turns, the magnets are rotated and the changing motion of the magnets creates magnetic flux that is then picked up by the HMC sensor.

The principles of operation for the devise are easy. The magnetoresistance is a function of $\cos(2\theta)$ where $\theta$ is the angle between magnetization $M$ and current flow in the sensor. HMC1512 is designed to be used in saturation mode, where the magnetization aligns in the same direction of the applied field. This happens when applied magnetic field is larger than 80 Oe (1 Oe = 100 µT). In this mode, $\theta$ is the angle between the direction of applied field and the current flow; the MR sensor is only sensitive to the direction of applied field. The sensor is in the form of a Wheatstone bridge with the same resistance for all resistors. The bridge power supply $V_S$ causes current to flow through the resistors, the direction as indicated in the figure for each resistor. HMC1512 has two identical MR bridges, coexisting on a single die. Bridge B physically rotates 45° from bridge A as shown in Figure5. The HMC1512 has sensor output $\Delta V=V_S S \sin (2\theta)$ for sensor A and sensor B output $\Delta V=-V_S S \cos (2\theta)$, where $V_S$ is supply voltage, $S$ is a constant, determined by materials. For Honeywell sensors, $S$ is typically 12mV/V. The typical output of both bridges is shown in Figure3. When bridge A and bridge B both output 50mV each, it indicates that the wheels rotate about 90° shown by yellow line. But when bridge A output 50mV while bridge B outputs -75mV, the wheels rotate around 15° shown by blue line. Thus, with bridge A and bridge B, the sensor covers the entire 90°.

Then, the data was sent to the computer through Arduino board. In other words, the sensor was supposed to detect any changes made by the wheel. However, since the input voltage was only 5 volts, the output voltage was too insignificant to be analyzed. Hence, the amplification of the input voltage was suggested and was later implemented.
Arduino Board

Arduino board basically acts as a platform that connects the physical sensors to the computer. Arduino board was used heavily to study the trends of the output before the oscilloscope was utilized. It was chosen over other microcontrollers because it is inexpensive and readily available in the market. It is very suitable for simple application and is also extensible to more complicated application. Besides, unlike most of the other microcontrollers, Arduino runs on multiple operating systems.
However, Arduino uses its own programming language based on a programming environment called Wiring. Although the programming language can be extended through C++libraries, the inconvenience caused is considered a major disadvantage. For example, Arduino does not output floating numbers which makes the digitization much more complicated than it already is. But other than that, Arduino serves perfectly well in our purposes on retrieving data from physical sensors.

In this research, particularly, the board was used in several applications. It was first used to retrieve acceleration data from dual-axes acceleration sensor, Memsic2125 and was later used to retrieve voltage data for magnetic sensor, HMC1512. Another advantage of this board is that it can be used for both analog and digital data. However, for all the sensors used in this research, only analog data was used. With this microcontroller, analysis was done based on the values output onto the computer. By that, we could tell how the vehicle moves.

**Coding**

The coding of the Arduino board was done in the early stage of this research. As mentioned in earlier section, Arduino board uses its own programming language. Hence, it is more complicated to code a sophisticated application which requires more coding functions. However, reference can be found easily in Arduino website which helped a lot in the coding process. In this research, coding was done only for two sensors: Memsic2125 and HMC1512. Since the output and the mathematical model for both sensors are different, the coding was done separately. A sample of the code can be found in Appendix A.

Coding for Memsic2125 was lengthier compared to HMC1512 due to the information given by the sensors. Memsic2125 uses time as the main factor for determining the acceleration of each axis and that requires multiple loops while HCM1512 gives the angle directly out of the sensor. This is another reason Memsic2125 was replaced temporarily with HMC1512.

**Amplification Circuit Design**

The HMC1512 produces an output range of +/-60mV around a median voltage of about 2.5V. This provides an obstacle in measuring the vehicle’s steering angle, since a change in millivolts is very difficult to measure accurately. To just amplify the output signal would not work if the Arduino Diecimila board was to be used, as it cannot read voltages above 20V. In order to read the output to a reasonable accuracy, the change in voltage had to be amplified, but not the overall voltage.

Designing a circuit of op-amps was the best solution to this task, since a very specific job was needed and a new design could be made to fit on the car without impairing any mechanical functions. The purpose of the circuit was to modify an input voltage with a +/-100mV range to an output with a range of 0-5V. The +/-100mV range was used in case the sensor would output any changes larger than 60mV or if noise or interference caused the output to rise.
The circuit was created with the combination of four components, each made with an op-amp and a configuration of resistors to produce the desired function. The key component of the circuit was a difference amplifier (see Component A in Figure 7). This component took the difference of the sensor output, nullifying the overall magnitude of the output voltage. In this circuit the resistors are all the same value, which is an amplification factor of 1. This means that the difference is output without any amplification. Now with a signal of +/-100mV, a second power source is used to input a voltage of 100mV to raise the sensor output to a range of positive voltages. An op-amp is used in Component B to stabilize the second power supply. It works as a feedback system to ensure that the voltage supplied is held at a constant 100mV, and is not affected by the current from other parts of the circuit. The two sources are summed together with another op-amp (see Component C). This combines to a positive signal with range 0-200mV. Component D is a non-inverting amplifier; it amplifies the signal and keeps the voltage positive. Ideally, an amplification of 25 is needed to raise the signal to 0-5V. The amplification is equal to value of the resistors:

\[
\frac{R_{13} + R_{14}}{R_{14}} \tag{Eq. 1}
\]

So by using a resistance of 10 Ohms and 270 Ohms for R13 and R14, respectively, an amplification factor of 28 is acquired. This provides a final output from the circuit of a 0-5.6V signal with an initial input of a -100mV to 100mV potential difference. And since the sensor does not output to those extremes, the actual output will more closely fit the initial goal of 0-5V.

Figure 7: Circuit Design
Hardware implementation

Referring to the result of the computer simulation as shown in Figure 7, the schematic in Figure 8 was drawn.

![Schematic for Op-amp circuit.](image-url)

With this schematic, all the components were soldered carefully onto a very small piece of PC board as shown in Figure 9. In the first trial, only one bridge was used. Our intention was to investigate the reliability of the sensor before putting everything together which requires much more effort. Then, two strong magnets were mounted under the vehicle using hot glue before the circuit was attached to the vehicle. Notice in Figure 11 that the gap between the sensor and the magnets is very small but not touching and the sensor is directly under the magnets. This is because we would like to obtain the strongest possible magnetic field to make sure that the sensor is in saturation mode. With that, the vehicle was finally ready to be tested.
Figure 9: Angle detection circuit.

Figure 10: Magnets mounted under the vehicle.

Figure 11: Final setting for HMC testing.
Potentiometer

Figure 12: Servo devise

Figure 13: Potentiometer

Figure 12 shows a servo devise used to control the steering wheels. The main components of this devise are a motor and a very lightly damped potentiometer. In order to monitor the direction of the steering wheels, the voltage output of the potentiometer was collected. This is because the potentiometer sitting inside the servo is actually a sensor that monitors the direction of the wheels. So, by knowing how much voltage flow through the potentiometer, we can tell the angle of the wheels with respect to the vehicle. Referring to Figure 13, there are three pins coming out of the potentiometer. The first and the last pins are for input voltage and ground respectively while the middle pin is in which the voltage flows out. Hence, the middle pin of the potentiometer was wired because it gives information about the wheels’ direction. After the pin was soldered and the servo was put back onto the vehicle, the vehicle was readied to be tested.

The process of analyzing the output voltage of the potentiometer was complicated. An oscilloscope was used to read both the input voltage that powered the vehicle and the output voltage that came out from the potentiometer. Meanwhile, two independent power supplies were used to power the vehicle and the potentiometer separately. The results were then observed directly from the oscilloscope. We expected to have two sinusoidal voltage waves overlapping each other. However, first few trials were failures because there was too much noise generated hence no clear signal was determined. The system was then improved by having separate grounds to the potentiometer and the input voltage. Finally, we obtained readable signal and the results were later used to determine the vehicle transfer function.

Function Generator for Frequency Response

The steering servo installed in the UGV described in this paper is controlled by a Pulse Width Modulation (PWM) input. In order to replicate that signal and vary its frequency, a square wave was needed. A square wave would provide an on/off signal to the servo that is similar to a PWM input, whereas something like a sine wave would create a continuous signal that would complicate the response.
To create a square wave with variable frequency two function generators were used. One outputting a sine wave, the other a sawtooth wave. By adjusting the DC offset and amplification of the two signals such that they both match, and feeding them into the two inputs of an op-amp, a square wave is created. Keeping the sine wave constant and adjusting the frequency of the sawtooth varies the frequency of the outputted PWM square wave. The sawtooth was chosen to drive the frequency since it had a more accurate frequency tuner. An example of this method, the intersective method, is seen in Figure14: when the sawtooth wave (blue) is less than the sine wave (green), the PWM signal is in high state (1), otherwise it is in low state (0). The magnitude of the PWM signal created was calibrated such that the high and low states caused the vehicle to turn its wheels to their maximum left and right angles. When fed into the system, frequency response could be analyzed for any specified frequency.

**Servo Response and System Identification**

To determine the transfer function of the servo, the UGV was fed the square wave input created by the sine and sawtooth function generators. The system was hooked up to an oscilloscope, and the magnitude and phase data was read directly from there. The oscilloscope was calibrated by centering both the input sawtooth function, which determined the frequency of the input square wave, and the output from the servo. The frequency of the sine function was varied from 2Hz to 14Hz. Frequencies below 2Hz were neglected, because the inaccuracy of the oscilloscope caused the measured output and input amplitudes to increase with frequency at extremely low values. After 14Hz, the magnitude of the output of the servo was so small that the oscilloscope could no longer zoom in enough to get an
accurate reading of the data. In addition to that, the noise generated by the servo became so significant that an exact reading of the magnitude could not be determined.

The magnitude of the servo output was measured on the oscilloscope for the range of input frequencies. The increase in magnitude at each data point was determined by calculating its percentage in terms of the first measured value, which was used as a reference point. The magnitude was then converted to decibels using the equation

\[ M = 20 \log M_V \quad \text{[dB]} \]  

Eq. (1)

where \( M_V \) is the servo output magnitude in mV.

The phase angle between the input and output was determined by displaying both functions on the oscilloscope, and measuring the time difference between their respective peaks. This phase was converted to an angle in degrees using the equation

\[ \phi = \phi_{sec} \times f \times 360^\circ \quad \text{[deg]} \]  

Eq. (2)

where \( \phi_{sec} \) is the phase in seconds, and \( f \) is the input frequency in Hz.

Using the values calculated by Equations 1 and 2, a Bode plot was created to represent the response from the servo.
From the experimental data shown in Figure 15, theoretical lines were created to determine the equation of the servo’s transfer function. From preliminary analysis of the servo and the data collected from the first experiment, there were already some expectations for the type and behavior of the transfer function. It was observed during the first experiment that the wheels of the car started to oscillate more forcefully at a frequency right around 10Hz, which would imply that a resonant frequency existed somewhere nearby. Examining the behavior of the plots in Figure 15, it seemed that the magnitude of the response stayed constant until the input frequency reached 3Hz, at which time it began to decrease at a rate of -20dB/decade. This continued until an input of 10Hz, when the response decreased at -60dB/decade.

Due to the difficulty of measuring the phase difference, a trend for the phase plot was much more challenging to determine. Since the magnitude response clearly showed changes at \( f = 3\) Hz and \( f = 10\) Hz, it was known that the changes in phase must also occur at these frequencies. From the
phase graph in Figure 15, it was concluded that the phase angle decreased by 90° at an input of 3Hz, and decreased another 180° at 10Hz.

Figure 16 shows the theoretical representation of the servo’s response as determined from the experimental data. This interpretation of the magnitude and phase data is consistent with the expected response of a system. At 3Hz, the magnitude of the response decreases by -20dB/dec, and the phase decreases by -90°, which is indicative of the presence of a pole at that frequency. At 10Hz, the magnitude decreases by an additional -40dB/dec while the phase drops by another -180°, implying that there is a complex pair at the input frequency of 10Hz. As stated, observations of the car’s motion
suggested that a resonant frequency existed at a 10Hz input, so the results gained from the Bode plot make sense.

From the Bode plot in Figure 16, the transfer function of the servo could be easily obtained with a few assumptions. The servo was assumed to be moderately damped since little to no overshoot was observed in Figure 15, and from the information in (Ogata 229)\(^5\), a value of 0.6 was chosen for the damping ratio, \(\zeta\), of the complex pair. Using the equation

\[
s^2 + 2\zeta\omega_n s + \omega_n^2
\]

Eq. (3)

the complex pair occurring at an input frequency of 10Hz was represented by

\[
s^2 + 12s + 100
\]

Eq. (4)

with \(\zeta = 0.6\) and \(\omega_n = 10\text{Hz}\). Using this, the model of the servo’s transfer function became

\[
G_s = \frac{1}{(s+3)(s^2+12s+100)}
\]

Eq. (5)

**Controller Design**

After determining the transfer function for the servo, a root locus was created to observe the behavior and stability of the system.

![Root locus for servo transfer function](image)

**Figure 17:** Root locus for servo transfer function, \(G_s\)
Figure 17 shows that for a certain gain $K$, the system will cross the Imaginary axis and become unstable. Using Matlab’s SISOTOOL, this $K$ was determined to be around 1660. Since it is unlikely that this large of a gain will ever be reached, the system is a fairly stable one.

The servo model $G_s$ is a Type 0 system, with no poles located at the origin. This means $G_s$ has a constant steady-state error for a unit step input. Using the equation

$$e_{ss,\text{step}} = \frac{1}{1 + \lim_{s \to 0} G_s(s)}$$

Eq. (6)

for a unit step input, the steady-state error was found to be $e_{ss} = 0.00332$, which is very close to zero. Since the inputs sent to the servo are square waves, the unit step error is the main type of error that needs to be considered. However, when the servo receives a series of commands for a continuous turning correction, the command may seem more like a unit ramp input, making it reasonable to also consider the effects of a unit ramp. For a Type 0 system, the steady-state error for a unit ramp input is infinite, as shown in the equation

$$e_{ss,\text{ramp}} = \frac{1}{\lim_{s \to 0} sG_s(s)}$$

Eq. (7)

Equation 7 shows that if the servo were to receive a command resembling a unit ramp, it would completely disregard the command. This created the need for an integral controller, and a PI-controller was designed to make the unit ramp steady-state error go to zero.

With the addition of the controller, the new transfer function for the open loop system became

$$\bar{G}_s = \frac{79.6801(s+.01)}{s(s+3)(s^2+12s+100)}$$

Eq. (8)
Figure 18 shows the root locus for the controlled system, $G_s$, which is represented by Equation 8. This system is more likely to go unstable, as a pole was added to the origin. Also, using SISOTOOL it was determined that a gain of $K = 19$ was needed to push the system unstable – a drastic change from the uncompensated system. Since the improvement to zero steady-state error for a unit ramp input may be desirable, the benefits and detriments of each option needed to be weighed before a decision can be reached. If the system gain can remain below $K = 19$, it would be beneficial to add the PI-controller to the servo model. If the servo were to completely ignore an input command, the goal of the project would not be met. Therefore, it can be concluded that it is worth accepting additional instability of the controlled system $G_s$ to ensure the maintained control of the vehicle.

**Heading Angle System Identification**

In order to design a controller for the heading of the vehicle, the affect of the steering angle on the actual system, the plant, had to be identified. The plant of the system was calculated by assuming a simplified model of the vehicle. A two-wheel bicycle model was used with a velocity applied by the back wheel and a steering angle applied by the front, as seen in Figure 19 below.
By separating the overall velocity, $V_F$, of the vehicle into body-frame components, $V_R$ and $V_T$, a relation for the steering angle can be found, (Eq. 9). This can be further simplified to (Eq. 10) by assuming the steering angle is small.

$$\tan(\delta) = \frac{V_T}{V_R}$$  \hspace{1cm} (Eq. 9)

$$\delta = \frac{V_T}{V_R}$$  \hspace{1cm} (Eq. 10)

The component $V_T$ is equal to the distance from the center of mass to the front wheel, $l$, times the angular velocity, where $\theta$ is the angle of the vehicle heading with respect to the inertial frame:

$$V_T = l \dot{\theta}$$  \hspace{1cm} (Eq. 11)

Substituting this into (Eq. 10) and solving for theta dot produces:

$$\dot{\theta} = \frac{\delta V_R}{l}$$  \hspace{1cm} (Eq. 12)

With (Eq. 12), it is now possible to relate the steering angle to the heading angle of the vehicle, and from there, design a system to control vehicle heading.
The overall objective for the steering controller is to allow a heading angle to be maintained. A heading command is input, $\theta_c$, and the system adjusts the steering such that the error, $\theta_{error}$, equals zero. The heading angle is altered by changing the steering angle, $\delta$, the angle of the wheels compared to the body of the vehicle.

To maintain a heading angle a controller, $K$, must be introduced into the system. The controller would be fed the error between the commanded heading and actual heading, $\theta_{error}$. The controller would most likely output a programmed steering angle command, based on a range of heading angle error. This conditional statement programming would provide an autopilot with minimal overshoot, but while still utilizing a less complex formula of step-inputs only.

The error in the heading angle is calculated by subtracting the output angle from the commanded angle for heading. The servo outputs a steering angle, which when multiplied by the plant, produces $\theta'$, as shown in (Eq. 12). With an integrator in the loop, the current heading angle, $\theta$, is output, providing the feedback necessary for the error calculation.

### Conclusion

System identification through frequency response provides an accurate method to model transfer functions for a mechanical vehicle. The results obtained correlated well with preliminary observations and expectations, as well as help visualize the stability of the system. While a controller was not needed for a stable steering system, its implementation allows for future modification of the heading controller to allow for stable ramp response.

Errors in measurements were mostly due to the limitations of the oscilloscope. The magnification of the sensor output was inadequate for higher frequencies, and reading the phase was especially difficult with an analog oscilloscope. In future experiments a digital oscilloscope with a freeze-frame feature would alleviate this problem significantly. Prolonged use of the servo and the potentiometer within it caused erratic behavior, mostly due to overheating of the component. This was avoided by disconnecting power to the servo periodically.

The HMC1512 magnetic sensor and amplification circuit will allow for a similar method of system identification to be applied to the speed controller. Another method that may work well for the
speed controller would be analysis of response within the time domain. Utilizing the data from the magnetic sensor of recorded wheel rpm, an exponential equation can be fitted to the response of various inputs. Due to the type of data recorded for this component of the vehicle, analysis within the time domain, as opposed to frequency response, may be more accurate and intuitive. However, this method lacks the advantages of frequency response where the data is already collected in a form easy to analyze. In order to identify transfer functions, a transformation to state-space representation is required.

References


Appendix A - Sample coding for Memsic2125

```c
// Pin declarations
int ledPin = 13;
int xAccelPin = 6;
int yAccelPin = 7;
int tempPin = 0;

// Instrument Settings
float dutyCycleGConversion = 0.125;

// Signal names
const char * xAccelName = "xAccel";
const char * yAccelName = "yAccel";
const char * tempName = "temp";

// Signal declarations
int xAccelSignal = HIGH;
int yAccelSignal = HIGH;
int tempSignal = HIGH;

// Variable Declarations
int xAccelCount = 0;
int yAccelCount = 0;
int tempCount = 0;

// Signal Presence Variables
boolean xAccelPresent = false;
boolean yAccelPresent = false;
boolean tempPresent = false;

// Output Variables
int xAccel = 0;
int yAccel = 0;
int temp = 0;

void initializePulseTrain( const char * name, int & signalPin, boolean & signalPresent) {
    // Declarations.
    int signal = HIGH;
    int count = 0;
    int countMax = 10000; // maximum wait for signal change

    // Wait for low signal.
    while(signal==HIGH && count<countMax) {
        signal = digitalRead(signalPin);
        count++;
    }

    // Wait for high signal.
    while(signal==LOW && count<countMax) {
        signal = digitalRead(signalPin);
        count++;
    }

    // Check for initialization.
    if (count==countMax) {
        Serial.println("Failed to initialize signal:");
        Serial.println(name);
        signalPresent = false;
    } else {
        Serial.println("Initialized signal:");
        Serial.println(name);
        signalPresent = true;
    }
    Serial.flush();
}

float findDutyCycle( const char * name, int & signalPin, boolean & signalPresent) {
    Serial.println("Calculating duty cycle.");
    int signal = HIGH;
    int countLow = 0;
    int countHigh = 0;
    int countMax = 10000; // maximum wait for signal change
    float dutyCycle = 0;
    float accel = 0;
    // Count number of high signals.
    while(signal==HIGH && countHigh<countMax) {
        signal = digitalRead(signalPin);
        countHigh++;
    }
    // Loop
    void loop() {
        if (xAccelPresent) findDutyCycle(xAccelName, xAccelPin, xAccelPresent);
        if (yAccelPresent) findDutyCycle(yAccelName, yAccelPin, yAccelPresent);
        if (tempPresent) findDutyCycle(tempName, tempPin, tempPresent);
    }

void setup() {
    // Initialize Serial.
    Serial.begin(9600);
    Serial.println("Serial connected.");
    Serial.flush();
}
```

```c
void loop() {
    if (xAccelPresent) findDutyCycle(xAccelName, xAccelPin, xAccelPresent);
    if (yAccelPresent) findDutyCycle(yAccelName, yAccelPin, yAccelPresent);
    if (tempPresent) findDutyCycle(tempName, tempPin, tempPresent);
}
```
// Count number of low signals.
while(signal==LOW && countLow<countMax) {
    signal = digitalRead(signalPin);
    countLow++;
}

// Calculate duty cycle.
dutyCycle = float(countHigh)/(countLow + countHigh);
accel = (dutyCycle-0.5)/dutyCycleGConversion;

// Output duty cycle or report error.
if (countLow>=countMax || countHigh>=countMax) {
    Serial.print("Signal fluctuation not detected for ":");
    Serial.println(name);
    signalPresent = false;
} else {

    // duty cycle
    Serial.print("Duty cycle of signal : ");
    Serial.print(name);
    Serial.print(" ");
    int dutyCyclePercent = 100*dutyCycle;
    Serial.print(dutyCyclePercent);
    Serial.println(" %");

    // acceleration
    Serial.print("Accel : ");
    Serial.print(name);
    Serial.print(" ");
    int accelMilli = accel*1000;
    Serial.print(accelMilli);
    Serial.println(" mg's");

    signalPresent = true;
}
Serial.flush();