Buzz Ball

EAA Founders Innovation Prize Entry

Ethan Brodsky

Initially Submission: June 15, 2017

Revised: July 25, 2017

Background

Spins subsequent to aerodynamic stalls are a major cause of fatal loss of control accidents. While most aircraft have predictable behavior when stalled in coordinated flight, uncoordinated stalls with the "ball out of center" are not necessarily so forgiving, and the resultant loss of roll control can lead to immediate and severe roll excursions, unexpectedly putting the aircraft in an extreme bank angle from which recovery may be difficult or impossible. This is especially dangerous when it occurs at low altitude. For example, a pilot attempting to tighten an overshooting base-to-final turn by using bottom rudder to skid the aircraft, while already imposing maneuvering loads in a steep turn, leading to an accelerated stall and a drop of the already-low wing and out-of-control impact with the ground, is a classic and too-often repeated occurrence in the NTSB accident database. While all low-altitude stalls are dangerous, recovery from a coordinated stall without an associated loss of roll control is far more likely.

It is well understood in aviation that this stall-spin scenario can be avoided by maintaining coordinated flight. While aviators since the beginning of flight have been reminded that they must use the rudder to do this, especially during slow flight, the high prevalence of stall-spin accidents among pilots of all experience levels after nearly 100 years of flight suggests that "keeping the ball centered" can be a challenging task.

Definition of Coordinated Flight

"Coordinated flight" is defined as flight without sideslip, or "sideways motion" that is not aligned with the relative wind in the vertical axis. Mathematically, it is a state in which the aircraft sideslip angle, or "directional angle of attack" (typically represented as β in flight dynamics equations) is zero. In addition to the improved behavior in stalls, zero sideslip angle is desirable because it minimizes drag and is more comfortable for passengers.

It is nontrivial to measure sideslip angle. In flight testing, it is typically measured using a vane-type transducer or a five-hole pressure probe, but such systems are not typically installed on small general aviation aircraft. Instead, lateral acceleration at the aircraft's center of gravity is used as a surrogate for sideslip angle. When an aircraft is flown in a slip, aerodynamic forces on the aircraft's side surfaces traveling into the relative wind will create an acceleration that is not aligned with the orthogonal aircraft axes. Zero lateral acceleration indicates zero sideslip (assuming there are no other external forces not aligned with the aircraft axes, such as thrust asymmetry in a twin-engine aircraft), and non-zero lateral acceleration is related to sideslip in a non-linear relationship that is based also on airspeed and aircraft weight. Coordination is typically indicated using a "turn

coordinator" or "turn and slip indicator" instrument, which includes a curved bubble level-style inclinometer (in some pusher or twin-engine aircraft a yaw string is used instead). These instruments provide a simple visual representation of coordinated flight, with a scale that relates to the lateral acceleration perceived by aircraft occupants. The inclinometer is typically marked with two vertical lines, sometimes called the "cage", that mark the central position of the ball in coordinated flight and allow quantification of how "far out" from center the ball is.

Motivation

It is difficult for many pilots to consistently connect the intellectual knowledge that maintaining coordination is important to safe flight with the physical sensory and motor processes required to actually do so. While many experienced pilots (especially those who fly taildraggers, gliders, or other aircraft with high adverse yaw moments) have developed a "seat of the pants" feel for required rudder inputs, for many pilots this intuitive sense never develops, and they must constantly remind themselves to check the coordination instrument and adjust their rudder pedal forces accordingly, a task that is often forgotten during high-workload stages of flight. The fact that coordination can only be checked visually imposes another task on the visual sensory system, which is often already task-saturated. This is especially true for VFR pilots who tend to be more accustomed to looking outside and have not developed a good instrument scan.

Pilots maneuvering visually at low altitude (e.g. for a pattern and landing) must only monitor two instruments to maintain controlled flight – the airspeed indicator and the turn coordinator (perhaps this is why the conventional six-pack panel layout places those two instruments in a vertical column). Eliminating the need to look at the turn coordinator would halve the number of instruments that the pilot must look at, and in an aircraft equipped with a glareshield-mounted or heads-up style AoA indicator or auditory/vibratory pre-stall warning system, the pilot would be able to keep his or her attention focused entirely outside.

The focus of this work is to eliminate the need for a pilot to visually check a coordination instrument by adding a system that gives coordination feedback using tactile (touch) presentation. While a pilot's visual sensory system is highly loaded during flight, the pilot's somatosensory system is very much underutilized. The somatosensory system includes the senses of touch, balance, warmth, pain, and body and joint position and strain (proprioception). The "sense of touch" itself includes a number of different sensations; our bodies are capable of sensing mechanical pressure, displacement of tissue, and vibration.

It is intuitive to present coordination indications to the somatosensory system, as it is the system that *should* be sensing coordination by perceiving lateral acceleration or forces transmitted to the body through the seat (much as the body can sense the G loads in a highly loaded turn).

Overview

For the purposes of the EAA' Founder's Innovation Prize competition, the *Solution* is the "Buzz Ball" system described below. The *Condition* is uncoordinated flight and the *Solution* addresses this condition by providing a means of increasing a pilot's ability to recognize and correct the condition, reducing the incidence of loss of control due to uncoordinated stalls and thus save lives.

The system consists of two physical parts. The pilot-facing portion is very simple and consists of a thin seat cushion overlay with two small embedded vibrating actuators, positioned such that one is centered under each of the pilot's buttocks. A prototype of the seat overlay is shown in Figure 1. During coordinated flight, the system is idle, but when the ball moves out of center, the actuator on that side vibrates. Instead of having to remember to look at a gauge and then "step on the ball", the pilot will be notified of the uncoordinated flight condition and can simply "step on the buzzing ball" to correct it.

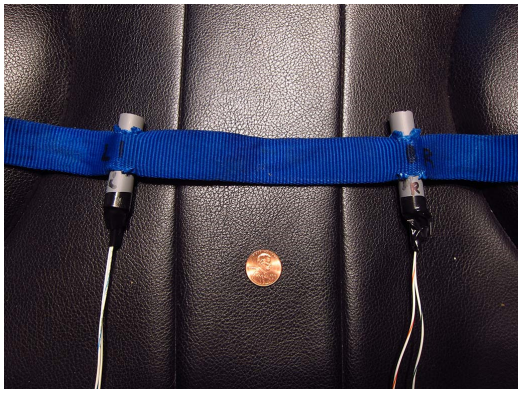

***Figure 1**: Prototype seat cushion overlay*

The second part of the system is the control computer, shown in Figure 2. This computer is a small package (about the size of two decks of playing cards) and must be mounted to a horizontal surface in the aircraft. It contains three stacked circuit boards – an Arduino-based computer board, a prototyping board with an inertial measurement unit (IMU), and a motor drive board. A single cable connects the two modules, and power is supplied using a small external battery or 5V USB-style power supply from the aircraft power.

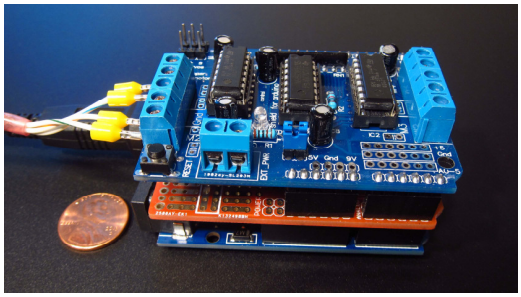

***Figure 2**: Control computer*

The system was tested in my aircraft, a 1981 Wag-Aero Wag-a-Bond Traveler, which is an experimental amateur-built replica inspired by the Piper Vagabond. Informed consent was obtained from all participants.

Hardware

The control computer is an HiLetgo-branded clone of the Arduino UNO R3 board (Figure 3). It has a Microchip ATmega328P processor, which is an 8-bit AVR microcontroller with 32 KB of programmable flash memory, 2 KB of RAM, and multiple channels of analog and digital input and output (IO). It is programmed by a laptop using a USB cable, and the Arduino developer community provides a free open-source development environment (Arudino IDE, from arudino.cc) that allows quick and easy development and deployment of code onto the hardware. The board has headers that allow "shield" boards to be stacked on top of it.

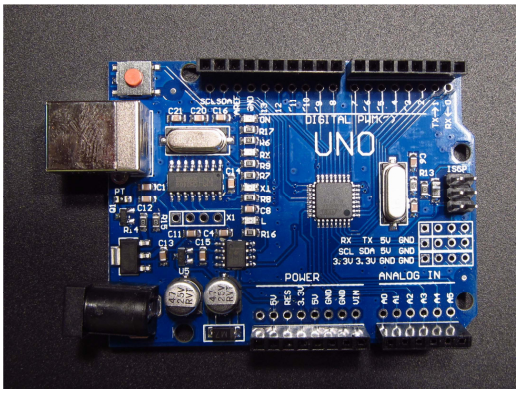

***Figure 3**: Computer processor board*

Stacked on top of the control computer board is a prototyping shield with a form factor that matches the UNO R3. The shield is attached using stackable pass-through headers, so an additional board can be placed on top of it. The only component on this board is the inertial measurement unit (IMU), an Adafruit 1120 triple-axis accelerometer/magnetometer board based on the STMicroelectronics LSM303DLHC sensor. The board is arbitrarily mounted such that the +X axis is aft in the aircraft, +Y is right, and +Z is up. The sensor chip communicates using the I²C serial

communication protocol, which is natively supported on the Arduino platform. Assembly of this board required soldering the stackable headers and IMU module into place, plus the addition of four jumper wires. The wires connect the GND, +5V, SCL, and SDA pins on the IMU module to the correspondingly labeled pins on the headers. The completed board is shown in Figure 4.

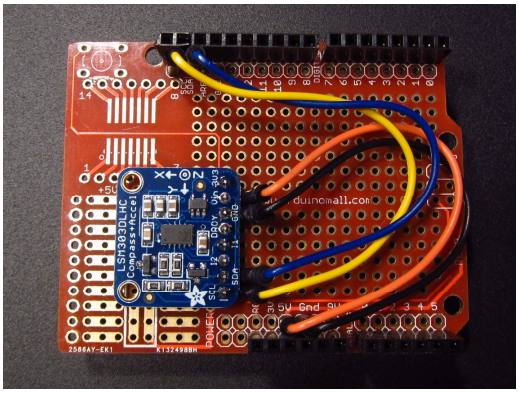

***Figure 4**: Inertial Measurement Unit Board*

Stacked on top is the motor control board, a Sainsmart clone of the "Adafruit V1 motor shield", shown in Figure 5. This board uses the TI L293D quadruple half-H driver chipset to provide four channels of DC motor control (or 2 channels of servo or stepper motors, functionality we do not use), with voltages up to 10V and currents up to 600 mA continuous and 1200 mA peak. The motor control interface uses most of the digital IO pins on the Arduino (pins 3-12).

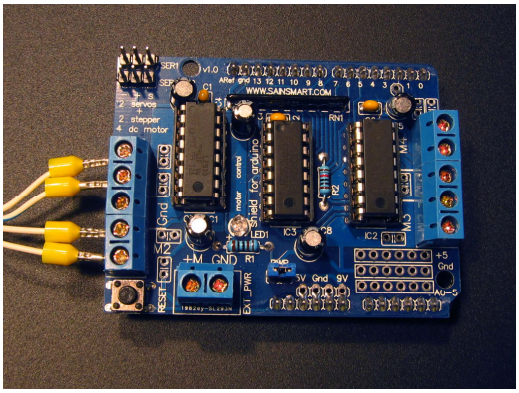

***Figure 5**: Motor driver board*

The control unit is connected to the seat overlay unit with a four-conductor shielded cable. Bootlace ferrules were crimped onto each conductor to secure into the screw terminals of the motor control board. The vibrator motor leads were soldered to the wires, then each connection was wrapped in electrical tape, then the pager motors themselves were wrapped in heat shrink (after removing the blue molded rubber shell), and the entire assembly was inserted into a ¼" ID x ½" OD x 2" L polypropylene standoff to protect the rotating eccentric weight from being touched. The polypro standoffs were then inserted into slits in a piece of 1" tubular webbing. The slits were spaced 6" apart, to approximately match the distance between the centers of the inventor's buttocks. The webbing was then secured to the existing aircraft seat using painter's tape, as previously shown in Figure 1.

A partial Bill of Materials can be found in Appendix A and wiring information in Appendix B.

Software

The software was developed in the Arduino C/C++-based integrated development environment and compiled using avr-g++. The software is quite simple and the full source code is included as Appendix C or can be downloaded from our source repository. Communication with the accelerometer is via the Adafruit_LSM303DLHC library, which is part of the Adafruit_Sensor library, both of which are available through the Arduino's standard library manager. Control of the motors is with an adaptation of public domain example code.

On startup, the software initializes the IMU and briefly pulses the vibration motors to verify their functionality. In the initial submission, the main loop sampled the accelerometer at 2 ms intervals and updated the system state at 50 ms intervals. To reduce sensitivity to airframe vibrations from the engine, digital low-pass filtering is performed. In the initial submission, this was achieved by averaging 25 samples over each 50 ms tick, then computing a rolling average over the preceding four ticks, so a total of 100 samples (200 ms) of lateral acceleration data are used. In the current

embodiment, the sampling occurs at 2.5 ms intervals and the system state is updated at 100 ms intervals, with the filtering is achieved using a first-order infinite impulse response (IIR) filter with a time constant of 250 ms.

The filtered lateral acceleration is then converted to "ball position" using an empirically calibrated gain (described below). If the ball position exceeds a predetermined threshold, the corresponding vibration motor for that direction is enabled. A small amount of hysteresis is applied to reduce rapid cycling due to sensor noise when ball position is very close to the threshold. For initial testing, the threshold was set such that the vibrations began when the ball was 3/8 of the way out of its cage, and stopped when it returned to within 0.370 of center.

Diagnostic information is output over the serial port (so a laptop can be attached for debugging and calibration) and an indicator LED is flashed at regular intervals.

The full source code can be downloaded from our source repository at the following URL:

https://bitbucket.org/ethan_brodsky/buzzball

Calibration

Figure 6: *Calibrating unit by jacking the aircraft to each ball position*

The instrument was calibrated with the aircraft stationary on the ground, in a process taking about fifteen minutes. With the aircraft parked on level ground such that the ball was centered, the lateral acceleration value is taken (using a laptop with a serial monitor), giving a zero value to compensate for the possibility that the cargo area deck is mounted unevenly relative to the airframe (though it may also include a small amount of error due to side-to-side inconsistencies in the landing gear or suspension). Then, one side of the aircraft was carefully lifted using a scissor jack under the axle stub, until the ball was one quarter of the way "out of its cage". The lateral acceleration reading was recorded at this point, and then process was repeated with the ball halfway out of its cage, three quarters of the way out of its cage, and fully out of its cage (inner edge of ball is exactly on white cage marking). This process was then repeated while lifting the other side of the aircraft. To get the ball fully out of its cage required lifting the wheel approximately 7 inches, for an aircraft with gear spaced approximately 70 inches apart, so the ball fully out of the cage corresponds to a stationary static bank angle of 6° (I am not certain if all coordination instruments have the same scale). Figure 6 shows the aircraft jacked such that the ball is fully out to the left. Lateral acceleration at these nine ball positions (centered, four to the left, four to the right) was then entered into a spreadsheet to calculate the gain and offset for the "lateral acceleration to ball position" transfer function. The entire calibration process takes only about fifteen minutes, and aside from the zeroing (which could be simplified to just a button push), may not be necessary in subsequent aircraft.

Installation

The solution was designed to be extremely easy to use and can be quickly installed either temporarily or permanently, in any aircraft, experimental or normal-category. It must merely be

affixed into place and requires no integration with any aircraft systems. A complete installation is shown in Figure 7.

The seat cushion is merely that, a thin cushion that overlays the existing cushion. It does not need to be attached to the aircraft in any way, though it may be held in place by Velcro, double-stick tape, or permanently installed inside the seat covering if desired.

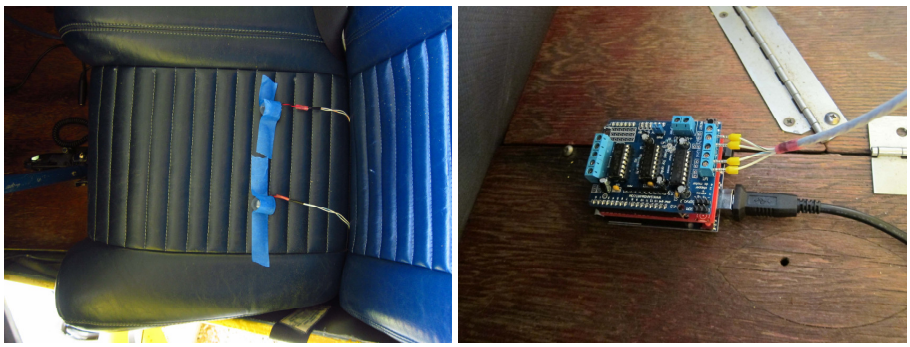

Figure 7: *Prototype was fully functional when installed in the aircraft using tape*

The control module must be attached to the aircraft in a fairly rigid manner, but it is small and light enough to fit almost anywhere. The only constraint is that it must be rigidly mounted to a horizontal surface, aligned with the longitudinal axis of the aircraft, and relatively close to the aircraft's center of rotation to avoid unwanted sensitivity to yaw rates. While the unit was hard-mounted to a solid structure in the test aircraft, large amounts of vibration from the engine suggest that mounting using an elastomeric isolation might be desirable – a small piece of foam in the mount and a few ounces of inertial mass bolted to the board was expected to be more than adequate.

Most of the testing prior to the initial submissions was performed with the unit duct-taped to the floor of the cargo area immediately behind the pilot's seat, but it could also be similarly installed on the floor, under the seat, or screwed to aircraft structure in an out-of-the-way location. Pushing it up against a bulkhead or transverse crossmember has proven sufficient to ensure accurate axis alignment, but if that was not possible, it would not be difficult to add a calibration feature that would allow arbitrary positioning (the calibration process would merely involve raising and lowering the tail while on the ground to distinguish between the forward/aft and left/right axes).

Since the initial submission, the mounting has been refined to reduce vibration. The control module was installed onto a damping mount designed for reducing vibration when using a GoPro-style camera on a small quadcopter UAV. These cost under $20 from Amazon and consist of two carbon-fiber boards separated by a four rubber dampers. The bottom board is screwed to a base which mounts in the aircraft, and the control module is attached to the top board with a single layer of 3M 300LSE double-sided adhesive foam tape. To further reduce vibration, 2 oz of additional mass was added to the control module using eight 0.25 oz stick-on steel wheel weights. In conjunction with the previously described software filter this was adequate such that spurious indications rarely occurred in the air, but this is an area in which further refinement would be useful. It is important also to constrain the wires in a manner such that they do not transmit vibration to the unit.

There is no limit on the length of cable between the control unit and the seat cushion. All control signals are low voltage and low current and present very low risk of fire, even if the wire is cut or short-circuited. The distance between the control module and the power supply is also practically unlimited, due to the small amount of power this device uses.

All testing was done powering the unit with a small "USB power pack", rated to 2200 mAh @ 5V DC. This battery is slightly larger than a pack of chewing gum, and any Seaplane Pilots Association member will recognize it as their "new member gift" last year. The system could equally well be powered by a 5V USB charger plugged into a cigarette lighter on the panel or a 5V supply hard-wired into the aircraft power. Note that a fuse is essential if any non-current-limited power source is used. The entire unit pulls on the order of 100 mA, so 10+ hours of operation is easily achievable off a small battery, or can run indefinitely if plugged into the ship's power.

Cost

This solution was intended to and has proved to be extremely affordable. The actual cost to build the first functional prototype was under $50, and that included the purchase of extra boards and vibrator motors that came in multiple-unit packs. All software was developed using free tools, though it does require a personal computer to program the module. It may cost slightly more for third parties to replicate my prototype as I used a number of items I already had on hand (jumper wires, multiconductor cable, webbing, USB battery). The only tools required are basic suppliers for soldering, crimping, and heat-shrinking of wire. Installation cost should be near zero, and it should be very easy to move between aircraft.

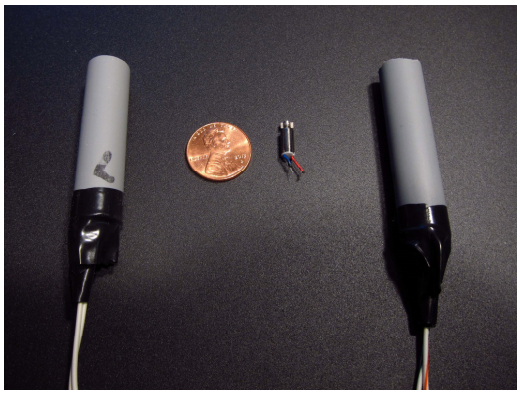

***Figure 8**: Pager vibration motor and assemblies*

Vibrating actuators

The most significant challenge in this project has been finding a suitable vibrating actuator to provide tactile feedback through the buttocks, which have proven to be fairly insensitive to touch with the body's weight compressing them.

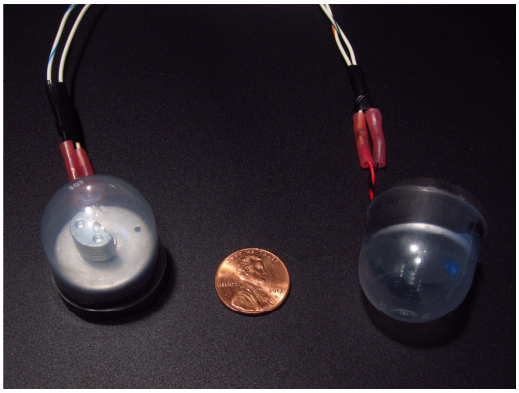

***Figure 9**: PS3 vibration motor assemblies*

The initial prototype used vibrating motors intended for use pagers or cell-phones (Figure 8). These motors were rated for 3500 RPM at 3.0VDC, pulling 70 mA continuous and 90 mA to start. They were overdriven at 5V, presumably running at around 5500 RPM and pulling around 120 mA. While they worked great for testing in an office chair and in an on-road vehicle, they were barely adequate to be felt in the high-vibration environment of a single-engine piston aircraft. This is especially true after the motor is constrained into a mount that holds it in place while still allowing the eccentric weight to rotate while underneath a seated passenger – this seems to substantially reduce the perceived vibrations. One should also note that these motors are of quite low and variable quality, and the speeds output varied substantially among the five that came in the pack I purchased. As these motors only used 1/5 the current-driving capacity that the motor controller board could handle, larger motors were tried.

A second prototype concept used the vibrating actuator from a Sony PlayStation 3 controller. A GameStop BB-6308 PS3-compatible controller was purchased at a garage sale (for $3) and disassembled to remove the vibrating motors. These were much larger motors than the pager units,

measuring 1" diameter x 0.5" length, and had large eccentric weights. They were of unknown specifications but were presumably designed to work on 3V based on the fact that the controller had 2xAA batteries. When run at 5V, it pulled around 20 mA and spun substantially slower than the pager motor. To protect the eccentric rotating weight, the motor unit was slipped into a "acorn" plastic capsule of the type found in gumball or claw-style novelty vending machines, as shown in Figure 9. Unfortunately, vibrations from this unit were even less perceptible than those from the pager motor. The author hypothesizes that touch receptors in the skin are much more sensitive to higher-frequency stimulus and that the more slowly rotating actuator is not suitable for this application.

A third prototype was constructed using a Portescap 16N28-207E DC brushed motor. It is rated for 10800 RPM at 12VDC and for loads of up to 240 mA, and was tested both at 12V (using an external power supply) and underdriven at 5V to retain the simplicity of a single power supply on the system. An eccentric rotating weight was constructed by cutting off two units of a ground terminal bar intended for use in an electrical service panel and using a screw to clamp it onto the shaft, as shown in Figure 10. Due to poor weather conditions and timing constraints, this assembly could not be tested in flight, but for ground-based testing, vibrations from this unit were extremely strong and easily perceptible at both 12V and 5V. Two factors counting against this unit, however, are that it is extremely expensive, costing $39 each in small quantities (so two of them would nearly triple the overall bill of materials), and that it gets quite hot, exceeding 120°F (50°C) after a few minutes of continuous running, which is well below the 100°C rating, but too hot for comfort to be sat upon without some insulation. The temperatures were much lower when unloaded, so it is likely due to the radial load the eccentric weight imposes on the shaft. This could be mitigated by setting software limits on continuous vibration – there is no reason the system needs to actuate continuously for several minutes while I am back-taxiing on the off-camber grass strip my aircraft is based at – it could instead be disabled after 30 s of continuous vibration and just buzz periodically. If these high-end motors are to be further investigated, a better choice might be the 207P, which is rated for 8000 rpm at 4.8V, or the 210E, rated for 9690 rpm at 7.5 V (6460 rpm at 5V), instead of underdriving the 207E at 5 V to get 4800 rpm, provided that the assumption that higher vibration frequencies are desirable for perceptual purposes is correct.

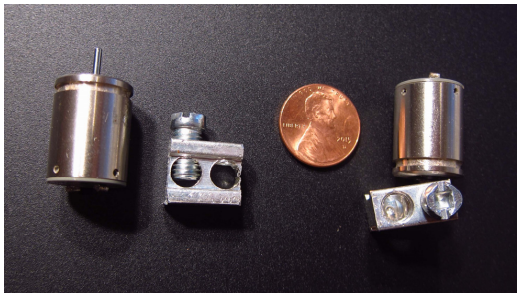

***Figure 10**: Escap Micromotor Vibration Unit*

The fourth prototype was constructed using what should've been used all along, a vibrating actuator intended for a similar operator-alert application. The General Motors 84017512 "Haptic Seat Motor" is one component of a "Lane Departure Warning System", part of the "Driver Alert Package" available as an option in a number of vehicles, including the 2017 Chevrolet Silverado. List price for the motors is $63.40 each and they are widely available for ~$40. These motors require 12V and pull approximately 100 mA and are extremely easy to feel. Two of these vibrating indicators were fit into slots cut in a foam "knee cushion", which served as an excellent seat overlay. The optimal positioning was found to be 9" apart, positioned fore/aft such that they were under the pilot's outer thighs. They are slightly less perceptible there than they are when in the original configuration centered under each buttock, but the wider spacing makes it easier to unconsciously distinguish which indicator is vibrating.

Further Study

While we have so far only undertaken the most basic testing of this system, it offers the potential for very in-depth analysis of a pilot's coordination performance. The computer in the control module is capable of measuring and logging lateral acceleration without triggering the seat vibrators, so it would be possible to compare a pilot's coordination with and without tactile feedback. After each flight, a report could be generated showing what percentage of time the ball was within certain tolerances of center. We hypothesize that the percentage of time spent in coordinated flight will be improved with the tactile feedback. It would also be interesting to break down the report into sub-categories such as "coordination performance in straight and level flight", "coordination performance while climbing" (with positive deck angle), and "while rolling left or right". As the IMU also measures heading, coordination performance "while in a stable standard rate left turn", "while in a steep turn", etc… could also be evaluated. The IMU used in this prototype was only a "six-axis" one, with a three-axis accelerometer and a three-axis compass, but for $5 more a nine-axis (these plus three-axis gyro) or ten-axis (those plus air pressure) could be used instead. As the only pilot who has tested this system so far is the author, I intend to loan it to a number of pilots of varying experience levels flying a variety of airplanes and record and assess their performance over time, with the hope that it will improve the coordination performance in all cases. Finally, it would be interesting to measure the performance of the system in VFR conditions with the lower half of the conventional turn coordinator instrument covered to remove visual feedback, forcing the pilot to rely only on tactile indication.

Some thought was given to varying the power level of the vibrating indicators or pulsing them periodically for various degrees of out-of-coordinated flight, but it was found that the buttocks are not particularly sensitive to slight variations in tactile simulation, so an all-or-nothing indication was chosen. It may be advantageous for the tactile stimulation to be more intense or to occur in a more sensitive area of the body, but the non-invasive simplicity of mounting the actuators in the seat led to this choice. Other options have been discussed, including auditory feedback using stereo tones over the headset, "heads-up" visual indicators in the panel, vibrating stimulation on the hands or other more sensitive parts of the body, as well as the possibility that such feedback could be used for other indications like course guidance during cross-country or instrument flight.

The original concept for this invention, conceived during a long flight accompanying a friend ferrying a Luscombe from Oregon to Wisconsin, was that the system would shock the pilot in a sensitive area to "punish" them for not maintaining coordinated flight, but for this initial proof of concept, we have opted for a less punitive means of training. Options to use a third indicating element to "reward" the pilot with positive reinforcement for maintaining coordinated flight have also been discussed, but not implemented.

Technical Notes

This section includes a number of technical notes which did not fit elsewhere.

It is important that IMU be mounted close to the aircraft's center of gravity. All testing was performed with the unit mounted approximately two feet behind the test aircraft's C_G. A mounting location far from the C_G means that angular acceleration (changes in yaw rate) will create lateral acceleration at that measurement location, with the direction depending on their position relative to the yaw center. This is unlikely to be a factor in most small general aviation aircraft, but one should note that it would be undesirable to put the sensing unit in the tail. Mounting locations away from the aircraft centerline laterally might also cause small measurement errors. This is likely

to be negligible in small GA aircraft, but mounting the sensor at the end of the wing would also be undesirable.

A design decision had to be made whether the unit should "normalize" its out-of-coordination quantification based on load factor. The physical inclinometer in the turn coordinator (a segment of a circular arc) will exhibit reduced sensitivity to fixed lateral acceleration at higher load factors, as it is measuring the direction of the net acceleration vector projected onto the plane of the instrument panel. To be representative of that, the software would have to normalize lateral acceleration based on the net acceleration vector (ignoring any longitudinal component). We have opted not to do that, which means that the sensitivity of the instrument to out-of-coordinated conditions will be greater at high load factors. If the vibrations start at "half a ball out" at 1 G, it will vibrate at "a quarter ball out" at 2 G. Whether this is the right choice or not is open to question, but my intuitive take on this is that coordination becomes especially crucial in highly loaded flight, so perhaps increased sensitivity here is appropriate. If someone feels otherwise, it would be easy to normalize by dividing a_y by $\sqrt{(a_y^2 + a_z^2)}$ or using $\tan^{-1}(a_y/a_Z)$ before further processing. One should note that the normalization and coordination situation becomes complicated when the aircraft is inverted, and this device has not been tested in that situation.

Conclusion

The BuzzBall was tested for several hours of pattern work and local flying in the author's aircraft and appears to satisfy its intended function, helping the pilot maintain awareness of aircraft coordination without having to look at an instrument. Moving forward, a more powerful tactile feedback actuator should be used, as it is essential that the pilot be able to perceive the vibrations and respond intuitively even when not paying attention to them, and the current vibrator is too weak to be consistently felt when distracted. The author intends to continue experimenting between now and Oshkosh. I invite any and all EAA members to build their own and test out or enhance upon this concept.

Appendix A: Bill of Materials

Initial Prototype (as submitted 2017-06-15)

Qty	Part	Vendor	Price
1	HiLetgo Arduino UNO R3 Board ATmega328P CH340 (w/USB cable)	Amazon	$6.99
1/3	Gikfun Prototype Shield DIY KIT for Arduino UNO R3 328P Ek1038 (pkg 3)	Amazon	$11.58
1	SainSmart L293D Motor Drive Shield for Arduino	Amazon	$5.50
1	Adafruit 1120 Sensor Development Tools Triple-axis Board - LSM303	Amazon	$17.49
2/5	Bluesky 5x DC3V/3500RPM Pager/Cellphone MicroVibration Motor 4x8mm	Amazon	$7.89
4	Jumper/hookup wires		
5 ft	M27500/22ML4T23 multiconductor cable		
4	Crimp bootlace ferrule (yellow, suitable for 22 AWG wire)		
6	3M TMW adhesive-lined polyolefin heat shrink (.183 and .255) precut lengths		
1 ft	1” Tubular webbing		
4 in	0.252” ID x 0.500” OD polypropylene unthreaded spacer stock	McMaster-Carr	

Current Embodiment (as of 2017-07-25)

Qty	Part	Vendor	Price
1	HiLetgo Arduino UNO R3 Board ATmega328P CH340 (w/USB cable)	Amazon	$6.99
1/3	Gikfun Prototype Shield DIY KIT for Arduino UNO R3 328P Ek1038 (pkg 3)	Amazon	$11.58
1	SainSmart L293D Motor Drive Shield for Arduino	Amazon	$5.50
1	Adafruit 1120 Sensor Development Tools Triple-axis Board - LSM303	Amazon	$17.49
2	General Motors 84017512 Haptic Seat Motor	GM dealer	$82.70
1	Fiskars 11x18x0.75” foam knee cushion	Home Depot	$5.97
1	GoolRC Gimbal FPV Camera Mount with Anti Vibration Plate for DJI	Amazon	$16.99
1	DROK Waterproof DC Buck Converter 8-22V to 1-15V 3A	Amazon	$9.96
1/10	iMBA-CCTV-PGTM Security Camera Power Plug Pigtail Cable (pkg 10)	Amazon	$5.49
1	RoadPro Fused Cigarette Lighter Plug with leads	Amazon	$3.88
5 ft	M27500/22ML4T23 multiconductor cable		
5 ft	Two-conductor power cable (18 AWG)		
	Crimp splices		
4	Crimp bootlace ferrule (yellow, suitable for 22 AWG wire)		
6	3M TMW adhesive-lined polyolefin heat shrink (.183 and .255) precut lengths		
8	0.25 oz stick-on steel wheel weights		

Appendix B: Wiring

Motor wiring:

L+ White
L- White/Blue
R+ White/Orange
R- White/Green

Pinout on motor driver board:

L M1 (+ outer, – inner)
R M2 (+ inner, – outer)

IMU to Arduino header on midboard:

3V3	n/c	
Vin	5V	Red
GND	Gnd	Black
DRDY	n/c	
I1	n/c	
I2	n/c	
SDA	SDA	Blue

Appendix A: Full source code

(as submitted, current version downloadable at https://bitbucket.org/ethan_brodsky/buzzball)

```
// buzzball
// June 2017
// Ethan Brodsky

// Copyright 2017 by Ethan Brodsky.  All rights reserved.

// Certain rights are available to EAA members in accordance with the 2016-2017 Innovation Competition rules
// see https://www.eaa.org/~/media/files/news/eaa%202016-2017%20fip%20rules%20final%20161013.pdf for details

#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM303_U.h>

#include "motor.h"

// Assign unique IDs to each sensor
Adafruit_LSM303_Accel_Unified accel = Adafruit_LSM303_Accel_Unified(54321);
Adafruit_LSM303_Mag_Unified mag = Adafruit_LSM303_Mag_Unified(12345);

void error_halt(void)
  {
   // shut off motors
    motor(1, RELEASE, 0);
    motor(2, RELEASE, 0);

   // flash LED slowly
    pinMode(LED_BUILTIN, OUTPUT);
    for(;;)
      {
        digitalWrite(LED_BUILTIN, HIGH);
        delay(1000);
        digitalWrite(LED_BUILTIN, LOW);
        delay(1000);
      }
  }

void setup(void)
  {
    Serial.begin(9600);
    Serial.println("Buzz Ball");
    Serial.println("");

    if(!accel.begin())
      {
        Serial.println("Error initializing accelerometer");
        error_halt();
      }

    if(!mag.begin())
      {
        Serial.println("Error initializing magnetometer");
        error_halt();
      }
```

```
    pinMode(LED_BUILTIN, OUTPUT);
    digitalWrite(LED_BUILTIN, HIGH);
  }

#define SAMPLE_PERIOD 2       // ms
#define TICK_PERIOD 50        // ms

#define TICKS_PER_DISPLAY 5
#define STARTUP_BUZZ_TICKS 40

#define SAMPLES_PER_TICK (TICK_PERIOD / SAMPLE_PERIOD)

#define ACCEL_ZERO 0.15
#define ACCEL_GAIN -0.86

#define BALL_THRESHOLD  0.375
#define BALL_HYSTERESIS 0.050

#define MOTOR_L 1
#define MOTOR_R 2

long sample = 0;
long tick   = 0;

bool buzz_L = 0;
bool buzz_R = 0;

float ay_prev1 = 0;
float ay_prev2 = 0;
float ay_prev3 = 0;
float ay_prev4 = 0;

float ay_buf[SAMPLES_PER_TICK];

long time_lasttick = 0;

void loop(void)
  {
    long time_sample_start = micros();

    sensors_event_t eventAccel;
    accel.getEvent(&eventAccel);

    ay_buf[sample++ % SAMPLES_PER_TICK] = eventAccel.acceleration.y;

    if (sample == SAMPLES_PER_TICK)
      {
        sample = 0;

        long time_thistick = micros();
        long ticktime = time_thistick - time_lasttick;
        time_lasttick = time_thistick;

        float ay_raw = mean(ay_buf, SAMPLES_PER_TICK);

       // average previous five values (1/4 s average)
        float a_y = (ay_raw + ay_prev1 + ay_prev2 + ay_prev3 + ay_prev4) / 5;
```

```
 ay_prev4 = ay_prev3;
 ay_prev3 = ay_prev2;
 ay_prev2 = ay_prev1;
 ay_prev1 = ay_raw;

 float ballpos = ACCEL_GAIN*(a_y - ACCEL_ZERO);

 float threshold_L = BALL_THRESHOLD - (buzz_L ? BALL_HYSTERESIS : 0);
 float threshold_R = BALL_THRESHOLD - (buzz_R ? BALL_HYSTERESIS : 0);

 buzz_L = (ballpos < 0) && (abs(ballpos) > threshold_L);
 buzz_R = (ballpos > 0) && (abs(ballpos) > threshold_R);

 if ((tick < STARTUP_BUZZ_TICKS) || buzz_L)
   motor(MOTOR_L, FORWARD, 255);
 else
   motor(MOTOR_L, FORWARD, 0);

 if ((tick < STARTUP_BUZZ_TICKS) || buzz_R)
   motor(MOTOR_R, FORWARD, 255);
 else
   motor(MOTOR_R, FORWARD, 0);

// diagnostic output on serial port
 if ((tick % TICKS_PER_DISPLAY) == 0)
   {
     sensors_event_t eventMag;
     mag.getEvent(&eventMag);

     float heading=(atan2(eventMag.magnetic.y,eventMag.magnetic.x)*180)/PI;
     if (heading < 0)
       heading = 360 + heading;

     float ticktime_ms = ticktime/1000.0;

     Serial.print(tick);
     Serial.print(" (");
     Serial.print(ticktime_ms);
     Serial.print(")    H: ");
     Serial.print(heading);

     Serial.print("   rawA: ");
     Serial.print(eventAccel.acceleration.x);
     Serial.print(" ");
     Serial.print(eventAccel.acceleration.y);
     Serial.print(" ");
     Serial.print(eventAccel.acceleration.z);
     Serial.print(" (ball ");
     Serial.print(ballpos);
     Serial.print(") L");
     Serial.print(buzz_L);
     Serial.print(" R");
     Serial.print(buzz_R);
     Serial.println();
   }
```

```
       // flash LED to verify code is still running
        int flash_index = tick % 20;
        if ((flash_index == 0) || (flash_index == 3))
          digitalWrite(LED_BUILTIN, HIGH);
        else
          digitalWrite(LED_BUILTIN, LOW);

        tick++;
      }

    long time_sample_end = micros();
    long time_sample = time_sample_end - time_sample_start;

    long delay_us = 0;
    if (time_sample < 1000*SAMPLE_PERIOD)
      delay_us = 1000*SAMPLE_PERIOD - time_sample;

    delayMicroseconds(delay_us);
  }

float mean(float buf[], int n)
  {
    float sum = 0;

    for (int ii = 0; ii < n; ii++)
      sum += buf[ii];

    return sum / n;
  }

// end
```

```
// Simple Motor Shield sketch
// ----------------------------------
//
// By arduino.cc user "Krodal".
// June 2012
// Open Source / Public Domain
//
// Using Arduino 1.0.1
//
// A simple sketch for the motor shield,
// without using the Adafruit library.
//
// The outputs can be used for DC-motors
// (either full H-bridge or just On and Off), lights,
// relays, solenoids, etc.
// But stepper motors can not be used !
// Servo motors can be used with the default Servo library.
//
// A maximum of 4 DC motors can be used with full-bridge,
// or a maximum of 8 normal outputs, or a combination.
// Two servo motors can always be used, they use the +5V
// of the Arduino board, so the voltage regulator could
// get hot.
//
// Tested with an Ebay clone with the Arduino Uno.
//
// Parts of the code are from an old Adafruit Motor Shield
// library, which was public domain at that time.
// This code is also public domain
//
// This simplified program is using the normal
// Arduino library functions as much as possible.
//
// The motors will make a whistling sound,
// due to the analogWrite() PWM frequency.
// The Adafruit library is specifically designed to avoid
// this, so use the Adafruit library for a better result.
//
//
//
// Connector usage
// ---------------
// The order is different than what you would expect.
// If the Arduino (Uno) board is held with the USB
// connector to the left, the positive (A) side is
// at the top (north), and the negative (B) side is
// the bottom (south) for both headers.
//
//   Connector X1:
//     M1 on outside = MOTOR1_A   (+) north
//     M1 on inside  = MOTOR1_B   (-)
//     middle        = GND
//     M2 on inside  = MOTOR2_A   (+)
//     M2 on outside = MOTOR2_B   (-) south
//
//   Connector X2:
//     M3 on outside = MOTOR3_B   (-) south
```

```
//     M3 on inside  = MOTOR3_A   (+)
//     middle        = GND
//     M4 on inside  = MOTOR4_B   (-)
//     M4 on outside = MOTOR4_A   (+) north
//
//
//         ------------------------------
//         | -+s                        |
//         | -+s                        |
//    M1 A |                            | M4 A
//    M1 B |                            | M4 B
//    GND  |                            | GND
//    M2 A |                            | M3 A
//    M2 B |                            | M3 B
//         |                      ..... |
//         ------------------------------
//                + -
//
//
//
// Pin usage with the Motorshield
// --------------------------------------
// Analog pins: not used at all
//     A0 ... A5 are still available
//     They all can also be used as digital pins.
//     Also I2C (A4=SDA and A5=SCL) can be used.
//     These pins have a breadboard area on the shield.
// Digital pins: used: 3,4,5,6,7,8,9,10,11,12
//     Pin 9 and 10 are only used for the servo motors.
//     Already in use: 0 (RX) and 1 (TX).
//     Unused: 2,13
//     Pin 2 has an soldering hole on the board,
//           easy to connect a wire.
//     Pin 13 is also connected to the system led.
// I2C is possible, but SPI is not possible since
// those pins are used.
//


#include <Servo.h>

void motor(int nMotor, int command, int speed);
void motor_output (int output, int high_low, int speed);
void shiftWrite(int output, int high_low);


// Arduino pins for the shift register
#define MOTORLATCH 12
#define MOTORCLK 4
#define MOTORENABLE 7
#define MOTORDATA 8

// 8-bit bus after the 74HC595 shift register
// (not Arduino pins)
// These are used to set the direction of the bridge driver.
#define MOTOR1_A 2
#define MOTOR1_B 3
```

```
#define MOTOR2_A 1
#define MOTOR2_B 4
#define MOTOR3_A 5
#define MOTOR3_B 7
#define MOTOR4_A 0
#define MOTOR4_B 6

// Arduino pins for the PWM signals.
#define MOTOR1_PWM 11
#define MOTOR2_PWM 3
#define MOTOR3_PWM 6
#define MOTOR4_PWM 5
#define SERVO1_PWM 10
#define SERVO2_PWM 9

// Codes for the motor function.
#define FORWARD 1
#define BACKWARD 2
#define BRAKE 3
#define RELEASE 4


// Declare classes for Servo connectors of the MotorShield.
Servo servo_1;
Servo servo_2;


void motor_setup()
{
  Serial.begin(9600);
  Serial.println("Simple Motor Shield sketch");

  // Use the default "Servo" library of Arduino.
  // Attach the pin number to the servo library.
  // This might also set the servo in the middle position.
  servo_1.attach(SERVO1_PWM);
  servo_2.attach(SERVO2_PWM);
}


void motor_loop()
{
/*
  // Suppose there are two servo motors connected.
  // Let them move 180 degrees.
  servo_1.write(0);
  delay(1000);
  servo_1.write(180);
  delay(2000);

  servo_2.write(0);
  delay(1000);
  servo_2.write(180);
  delay(2000);


  // Suppose there is a relay, or light or solenoid
```

```
  // connected to M3_A and GND.
  // Note that the 'speed' (the PWM, the intensity)
  // is for both M3_A and M3_B.
  // The output is a push-pull output (half bridge),
  // so it can also be used to drive something low.
  // The 'speed' (the PWM, the intensity) can be set
  // to zero, that would make the output disabled
  // and floating.
  motor_output(MOTOR3_A, HIGH, 255);
  delay(2000);
  motor_output(MOTOR3_A, LOW, 255);
*/

  // Suppose a DC motor is connected to M1_A(+) and M1_B(-)
  // Let it run full speed forward and half speed backward.
  // If 'BRAKE' or 'RELEASE' is used, the 'speed' parameter
  // is ignored.
  motor(1, FORWARD, 255);
  delay(2000);
  // Be friendly to the motor: stop it before reverse.
  motor(1, RELEASE, 0);
  delay(500);
  motor(1, BACKWARD, 128);
  delay(2000);
  motor(1, RELEASE, 0);
}


// Initializing
// ------------
// There is no initialization function.
//
// The shiftWrite() has an automatic initializing.
// The PWM outputs are floating during startup,
// that's okay for the Motor Shield, it stays off.
// Using analogWrite() without pinMode() is valid.
//


// ---------------------------------
// motor
//
// Select the motor (1-4), the command,
// and the speed (0-255).
// The commands are: FORWARD, BACKWARD, BRAKE, RELEASE.
//
void motor(int nMotor, int command, int speed)
{
  int motorA, motorB;

  if (nMotor >= 1 && nMotor <= 4)
  {
    switch (nMotor)
    {
    case 1:
      motorA   = MOTOR1_A;
      motorB   = MOTOR1_B;
```

```
      break;
    case 2:
      motorA   = MOTOR2_A;
      motorB   = MOTOR2_B;
      break;
    case 3:
      motorA   = MOTOR3_A;
      motorB   = MOTOR3_B;
      break;
    case 4:
      motorA   = MOTOR4_A;
      motorB   = MOTOR4_B;
      break;
    default:
      break;
    }

    switch (command)
    {
    case FORWARD:
      motor_output (motorA, HIGH, speed);
      motor_output (motorB, LOW, -1);     // -1: no PWM set
      break;
    case BACKWARD:
      motor_output (motorA, LOW, speed);
      motor_output (motorB, HIGH, -1);    // -1: no PWM set
      break;
    case BRAKE:
      // The AdaFruit library didn't implement a brake.
      // The L293D motor driver ic doesn't have a good
      // brake anyway.
      // It uses transistors inside, and not mosfets.
      // Some use a software break, by using a short
      // reverse voltage.
      // This brake will try to brake, by enabling
      // the output and by pulling both outputs to ground.
      // But it isn't a good break.
      motor_output (motorA, LOW, 255); // 255: fully on.
      motor_output (motorB, LOW, -1);  // -1: no PWM set
      break;
    case RELEASE:
      motor_output (motorA, LOW, 0);  // 0: output floating.
      motor_output (motorB, LOW, -1); // -1: no PWM set
      break;
    default:
      break;
    }
  }
}


// --------------------------------
// motor_output
//
// The function motor_ouput uses the motor driver to
// drive normal outputs like lights, relays, solenoids,
// DC motors (but not in reverse).
```

```
//
// It is also used as an internal helper function
// for the motor() function.
//
// The high_low variable should be set 'HIGH'
// to drive lights, etc.
// It can be set 'LOW', to switch it off,
// but also a 'speed' of 0 will switch it off.
//
// The 'speed' sets the PWM for 0...255, and is for
// both pins of the motor output.
//   For example, if motor 3 side 'A' is used to for a
//   dimmed light at 50% (speed is 128), also the
//   motor 3 side 'B' output will be dimmed for 50%.
// Set to 0 for completelty off (high impedance).
// Set to 255 for fully on.
// Special settings for the PWM speed:
//    Set to -1 for not setting the PWM at all.
//
void motor_output (int output, int high_low, int speed)
{
  int motorPWM;

  switch (output)
  {
  case MOTOR1_A:
  case MOTOR1_B:
    motorPWM = MOTOR1_PWM;
    break;
  case MOTOR2_A:
  case MOTOR2_B:
    motorPWM = MOTOR2_PWM;
    break;
  case MOTOR3_A:
  case MOTOR3_B:
    motorPWM = MOTOR3_PWM;
    break;
  case MOTOR4_A:
  case MOTOR4_B:
    motorPWM = MOTOR4_PWM;
    break;
  default:
    // Use speed as error flag, -3333 = invalid output.
    speed = -3333;
    break;
  }

  if (speed != -3333)
  {
    // Set the direction with the shift register
    // on the MotorShield, even if the speed = -1.
    // In that case the direction will be set, but
    // not the PWM.
    shiftWrite(output, high_low);

    // set PWM only if it is valid
    if (speed >= 0 && speed <= 255)
```

```
    {
      analogWrite(motorPWM, speed);
    }
  }
}


// ----------------------------------
// shiftWrite
//
// The parameters are just like digitalWrite().
//
// The output is the pin 0...7 (the pin behind
// the shift register).
// The second parameter is HIGH or LOW.
//
// There is no initialization function.
// Initialization is automatically done at the first
// time it is used.
//
void shiftWrite(int output, int high_low)
{
  static int latch_copy;
  static int shift_register_initialized = false;

  // Do the initialization on the fly,
  // at the first time it is used.
  if (!shift_register_initialized)
  {
    // Set pins for shift register to output
    pinMode(MOTORLATCH, OUTPUT);
    pinMode(MOTORENABLE, OUTPUT);
    pinMode(MOTORDATA, OUTPUT);
    pinMode(MOTORCLK, OUTPUT);

    // Set pins for shift register to default value (low);
    digitalWrite(MOTORDATA, LOW);
    digitalWrite(MOTORLATCH, LOW);
    digitalWrite(MOTORCLK, LOW);
    // Enable the shift register, set Enable pin Low.
    digitalWrite(MOTORENABLE, LOW);

    // start with all outputs (of the shift register) low
    latch_copy = 0;

    shift_register_initialized = true;
  }

  // The defines HIGH and LOW are 1 and 0.
  // So this is valid.
  bitWrite(latch_copy, output, high_low);

  // Use the default Arduino 'shiftOut()' function to
  // shift the bits with the MOTORCLK as clock pulse.
  // The 74HC595 shiftregister wants the MSB first.
  // After that, generate a latch pulse with MOTORLATCH.
  shiftOut(MOTORDATA, MOTORCLK, MSBFIRST, latch_copy);
```

```
  delayMicroseconds(5);    // For safety, not really needed.
  digitalWrite(MOTORLATCH, HIGH);
  delayMicroseconds(5);    // For safety, not really needed.
  digitalWrite(MOTORLATCH, LOW);
}

// end
```