5-2009

Automated flight through inertial navigation and digital fly-by-wire systems

Aaron French
University of Arkansas, Fayetteville

Automated Flight through Inertial Navigation and Digital Fly-by-Wire Systems

An Honors Thesis submitted in partial fulfillment
of the requirements of Honors Studies in
Mechanical Engineering

By

Aaron French

Spring 2009
Mechanical Engineering
College of Engineering
The University of Arkansas

Acknowledgements

Dr. Adam Huang: Thank you for allowing my assistance on this project.
Dr. Gay Stewart: Thank you for pushing me through two honors degrees, and keeping me sane.
Dr. John Stewart: Thank you for getting me through E&M and Quantum Mechanics.
Dr. Mary Jo Schneider: Thank you for taking the time out of your busy schedule to serve on my defense committee.

Table of Contents

Background

With technology advancing by leaps and bounds by the year, the uses of Unmanned Aerial Vehicles are becoming more apparent. The possibilities for an aerial vehicle that is fully autonomous are endless. Such aerial vehicles could divert manpower to humanitarian efforts, airlift, and other non-attack operations. Leaders can put their focus elsewhere on the battlefield with the knowledge that the skies over their heads will be well guarded. These desires give life to projects such as this. No advancement in civilization has ever been made without the collective revolutionary research of great minds, trial by fire, and daring to dream.

Extensive study of the modern Air Force reveals that modern tactical combat aircraft are intentionally built to be aerodynamically unstable so as to give an added degree of maneuverability. However, these unstable tendencies make the pilot's purpose much less achievable as he or she would be incessantly fighting the aircraft to maintain even the most basic modes of flying: straight and level flight. The digital fly-by-wire system, first pioneered by NASA, utilizes an onboard digital computer integrated into the flight controls to receive and interpret inputs from the pilot, and actuate the controls in such a way that they do what the pilot desires while the computer maintains stability on all axes. Without the onboard computer, most of today's fighter aircraft would be impossible to fly by human control alone. [10]

Currently, there are inertial navigation units and fly-by-wire systems available on the market in pre-assembled units that are not only large and heavy, but quite costly to the consumer. [6, 10] One of the aims of this project was to construct and use a unit

from recently available miniature flight control and navigation components (accelerometers, gyros, magnetometers, pressure altimeters, and GPS). The unit would be smaller, weigh less, and cost considerably less than its pre-assembled counterparts and should maintain the same relative degree of accuracy and precision. As discussed below, there are still some work left to complete this project as a fully viable prototype, and further work will be necessary to implement the prototype. Continued work to develop a new class of Unmanned Aerial Vehicle (UAV) with the capability of highly dynamic maneuvers is still being extensive researched around the country.

The other consideration is the need for onboard real-time tracking and correcting for an automated flight system when GPS should be more than adequate. In actuality an onboard real-time tracking system would provide better support for the autonomous control system in between GPS cycles. The GPS sweep updates roughly each second, whereas an onboard system can be set to cycle in milliseconds or even microseconds. Plus, with military uses, an aircraft would have to worry about signal jamming while in enemy territory. If anything ever went wrong with signaling to the GPS satellites or if the system ever went down, a redundancy such as an onboard tracking and control system would be necessary for the safety of the aircraft.

Objectives Tasked

Objective 1: Construct Components to be Integrated into Test Aircraft.
- **Task 1.1:** Purchase accelerometer, GPS, magnetometer, gyros, etc.
- **Task 1.2:** Construct control circuit (Dr. Huang).
- **Task 1.3:** Write programmable logic to carry out experimental tasks.

Objective 2: Construct Control Circuit Codes for Use on Test Aircraft.
- **Task 2.1:** Brainstorm methods for accelerometer algorithms.
- **Task 2.2:** Select and construct the best accelerometer algorithm.
- **Task 2.3:** Use accelerometer algorithm to construct rate gyro algorithm.
- **Task 2.4:** Simulate and troubleshoot accelerometer algorithm using function generator.
- **Task 2.5:** Simulate and troubleshoot rate gyro algorithm using function generator.
- **Task 2.6:** Implement control codes into circuit and run flight test.

Objective 3: Calibrate Automated Control System.
- **Task 3.1:** Alter weight distribution on test aircraft to calibrate three-dimensional stability.
- **Task 3.2:** Calibrate three-dimensional automated flight control with test routes.

Objective 4: Integrate with GPS and implement system.
- **Task 4.1:** Integrate and calibrate GPS.
- **Task 4.2:** Engage system to perform pre-specified routes and tasks/maneuvers.

Objective 5: Data Analysis, Final Report, and Presentation.
- **Task 5.1:** Acquire necessary field data.
- **Task 5.2:** Prepare report and give final presentation of results.

The above objectives detail what need to be done to develop a fully autonomous system. The scope of this project will only deal with objectives 1 and 2. The measure of success will be based upon the ability of the developed algorithm to successfully integrate a sinusoidal wave function fed as an accelerometer reading into the microprocessor by a function generator.

Objective 1-Component Construction and Programming

All of the components on board the aircraft can be categorized into three systems: sensors, central computer, and actuators/effectors. [10] The sensors interpret data from the aircraft's surroundings and the actuators/effectors; send it to the central computer, and the central computer sends signals back to the actuators/effectors to make any necessary changes. The central computer in this case is a Programmable Interface Controller (PIC) 16F877. [12] The pin diagram can be found in Figure B1 of Appendix B.

Just as in common appliances and household wares, sensors are present in just about every aircraft today, regardless of whether or not they have a digital fly-by-wire system onboard. The sensors employed in the test aircraft will provide similar flight information as the sensors in today's modern aircraft. The pressure transducers and sensors will take real time data to monitor altitude as well as rates of climb and descent. They operate by changing the small displacement of a diaphragm due to changes in pressure into an electrical voltage that will be interpreted into an altitude by the microcontroller. [5,11]

The accelerometer/inclinometer measures acceleration changes in the aircraft to give the microcontroller an idea of the aircraft's three-dimensional position relative to the ground, and the magnetometer assists the accelerometer by use of the Earth's magnetic field. [1,3,11] As the aircraft moves, inertial forces act upon it any time it changes direction or airspeed, as per Newton's Laws of Motion. These laws state that whenever an object in an inertial reference frame is in motion or at rest, it will stay in

motion or at rest until acted upon by some outside force. When the aircraft is stable along a particular flight path, any control inputs will disturb that equilibrium, much like turning a car to the left moves the car to the left, making the passengers, who tend to stay in straight-line motion by the above law, experience a feeling of being pushed to the right. Relative to the car, which is an accelerated reference frame, the passengers experience a "fictitious force" which pushes them to the outside of the turn. These forces due to the non-inertial frame move a tiny suspended object inside the accelerometer's circuit. This motion creates a voltage that is sent to the output of the device and ultimately to the microprocessor. [1,3] The codes then interpret this voltage output into a measurable acceleration, which can then be used in an algorithm to calculate an instantaneous position and velocity using some form of numerical integration. The method employed in the algorithm of this project was an averaging rule. This is a recursive method that uses two acceleration data readings to numerically integrate an instantaneous velocity. After at least two velocity readings are calculated, an instantaneous position can be calculated using the same technique.

The rate gyros measure the rotational velocities of the aircraft about the three coordinate axes. These gyros are tiny spinning discs that use gyroscopic principles and inertial principles commonly found throughout physics to measure the rate at which their orientation about an axis is changing, which is sent to the output of the device as a voltage, similar to the accelerometer. As the disc spins at a high rate of rotation, the inertial gyroscopic tendencies cause it to resist rotation about its axis. The relative position of the disc to its casing is how the device can interpret the three-dimensional

orientation of the aircraft. [2] A diagram of the rate gyros can be found in Figures B7 and B8 in Appendix B.

The first great undertaking of the project began in the fall, which was to learn the microcontroller's operating language: PIC Basic. This language is different than JavaScript, VB, or any other common programming language, which perpetuated a necessity to learn and test it. This was accomplished through the use of the PIC Basic Pro Manual, and running test codes to program working functions such as serial communications, analog-to-digital conversions, and onboard clock manipulations. [2] Samples of these test codes can be found in Codes C4 and C5 of Appendix C.

Objective 2-Control Code Construction and System Integration

The process of the accelerometer algorithm involves connecting the accelerometer to the appropriate pins on the microcontroller as defined in the codes. A reading is then taken from the accelerometer and put through an Analog-to-Digital Converter (ADC) on board the microcontroller. This converts data from the analog world, better known as the physical world, into digital data that can be processed more quickly by computers, without taking up as much storage space on the processor. From there, the algorithm stores the voltage readings as bit integers, which are then run through the recursive averaging rule to numerically integrate a velocity.

After the first couple of calculations are made, the algorithm sets up the same steps to be repeated in a continuous loop. At the end of each loop of calculations, the first values are thrown away and the more recent values become the first values in order to use the new readings for the next loop of calculations. This is how a recursive relation inside a loop of code operates, and is illustrated in the accelerometer codes in Code C1 of Appendix C. After the appropriate calculations are completed, the resulting integers are displayed to the ground station communicating with the aircraft through a debug command. From there, a separate algorithm on the ground station runs a conversion from bits to volts and from those volts to the appropriate units of acceleration, velocity, or position as necessary. [12,9] A logic diagram of the employed algorithm can be found in Figure B9 of Appendix B.

Once the codes were written, testing was conducted to ensure proper pauses and time intervals were in place. Since the recursive relation in the sample code uses a

fixed time interval, dt, it was necessary to know the exact amount of time needed for the microprocessor to run the loop each time. To remedy this, HIGH and LOW codes were temporarily entered into the loop to turn up the voltage in an unused pin at the beginning of the loop, and then turn it off halfway through the loop. An oscilloscope was then connected to the unused pin to display the voltage changes temporarily placed in the code. This registered as a square wave function on the oscilloscope, which made finding the period quite simple. This period was the representation of the actual amount of time taken to run the loop for each pass. Additional pauses were then inserted into the code to round the total time needed to an even 100 milliseconds, and the HIGH and LOW codes were removed.

The same methods that were used in the accelerometer algorithm were employed with the rate gyros as well. In this case, the output of the device was the instantaneous rotational velocity, which was then numerically integrated once to convert to the instantaneous rotational position. The codes for the rate gyros can be found in Code C2 of Appendix C. So between the accelerometer and rate gyro readouts, the aircraft operator would have instantaneous real-time information of the aircraft's acceleration, position, velocity, orientation in three dimensions, and the rate at which that orientation was changing. This is just like the airspeed indicator and attitude indicator in today's modern aircraft, but with higher rate of update, and at lower weight. While the aircraft instruments provide information to a pilot in the cockpit, these components provide the information to a remote ground station, which is more pertinent to the setup of the research.

The test aircraft to be used with this project was a simple remote-controlled prototype aircraft provided by the Design Build and Fly (DBF) team. It was a conventional tail-dragger design with a conventional tail section, which many tests showed would have the best performance with the least amount of drag. The aircraft was constructed using a special type of composite wrapped in a polymer coat that would reinforce its strength amidst vibrations in the air, and had two composite arrow shafts supporting the length of the fuselage. This aircraft was then mounted with an electric motor and propeller and was ready to fly.

Once completed, the Design Build and Fly team would relinquish the aircraft while embarking upon the construction of their competition plane. The test aircraft would then be fitted with the control circuit board constructed by Dr. Huang, containing the necessary components needed for the first phase of flight testing: position and attitude tracking. This was to be accomplished by programming the aforementioned coding into the microprocessor, or flight computer, and linking it to a ground station via radio communication. The ground station would be equipped with imaging software to show the aircraft's position and attitude in three-dimensional space. If successful, this would show the codes and equipment to be fully operational and ready for the second of flight testing: autonomous three-dimensional stabilization.

The selection of a proper constant time interval is still essential for the proper functioning of the algorithm. The averaging method operates by taking the average of acceleration values provided by the analog-to-digital converter and multiplying it by the constant time interval that occurs between the two accelerations. The selection of the

length of the time interval directly affects the amount of error present in the data. The physics behind signal processing teaches that a higher sampling rate contains less error within the resulting data. However, under one of the previously mentioned methods, more accuracy would have been accompanied by reaching the maximum value of 1023 much quicker, thus limiting how far the aircraft could go in the physical world and still be tracked. The averaging method only uses two acceleration points at a time. After performing the necessary calculations, the recursive aspect throws away the oldest value and shifts the most recent value in its place. The newest value then becomes the second value needed to run the algorithm on the next loop pass. With this method, the only time interval limitation is the processing capability of the analog-to-digital converter. A small amount of time is needed between readings to allow the converter to reset itself in preparation for the next reading. That, reset time is the only limit on how fast the sampling rate can be and thus, how small the time interval can be.

Once the best method was chosen, it had to be employed, simulated with test codes and functions, and rewritten many times to ensure proper syntax and logical paths were followed, allowing for any possible errors that could arise within the system while measurements were being taken. As is customary of testing components and parts of aircraft in the design and construction phase, these extensive simulations give the best idea possible of how components or codes will behave in the real world by running them in a controlled lab environment. Fortunately, after much troubleshooting and many extensive simulations, the code is hypothesized to be successful when put into the aircraft for flight testing. To simulate this, a function generator was connected

to the microprocessor instead of an actual accelerometer. This function generator then fed a simple oscillating sine wave to the microprocessor and the accelerometer algorithm showed the aircraft oscillating in its flight path and velocities. The rate gyro codes were tested in much the same way.

Lessons Learned and Results

The hardest part of anything involving computer programming is being able to articulate an idea or concept in terms a computer would understand and be able to execute. The next most difficult task is the extensive troubleshooting involved. When the code is tested and the results are different from what they should be, where does one begin to search for the problem? One of the most valuable lessons learned on this project is organization and proper "book-keeping" when writing code. This is necessary for searching through lines of code for the one punctuation mark that is out of place. Mastery of an entirely new system of code that was completely different from any previous programming experiences was required. As a result, I gained new appreciation for the extensive knowledge and skill required for complex computer programming.

Aside from an appreciation for programming, I learned many other valuable lessons. For example, I learned proper ways to construct many different circuits. Although circuits have a firm root in Electrical Engineering, many physics considerations come into play. The incorporation of capacitors in conjunction with an oscillator was necessary to override the onboard oscillator in the microprocessor to allow faster processing speeds. The oscillator system on board the microprocessor consists of an oscillator and two capacitors. The system was duplicated outside the microprocessor in the connected circuits to override the onboard oscillator with one five times faster. The functions performed in the tracking codes would have taken much longer at the default speeds on the microprocessor, which would ultimately lead to less accurate readings by the analog-to-digital converter when accounting for a constant time interval, dt. I also

had several practical uses of signal processing skills from optional circuit additions such as the oscillator to noise moderation and reduction monitored by the oscilloscope.

Simulations that were run on the algorithm proved to be successful. The function generator sent simulated raw data to the microprocessor for analysis and successfully showed the aircraft's position about all three coordinate axes as well as the rates of rotations and in which direction within a few tenths of an inch. The only thing that remains for completing the simulations is to put the data into imaging software such as LabView, and map a visual representation of the plane's flight path and orientation.

Continued Research

The next step for continued research is the completion of Objectives 3 and 4. Objective 3 is the first step towards automation, and uses Digital-to-Analog conversion to take inputs from the microprocessor and use them as analog manipulations of the flight controls to achieve a desired control input. The first step would be to position weights on the aircraft in a purposeful attempt to place the center of gravity of the aircraft in an unstable condition. Under normal circumstances, this would cause the aircraft to be extremely difficult to control for a human pilot and could eventually lead to an unsafe end to the flight. However, this is how many acrobatic and attack aircraft are configured, which gives them a higher degree of maneuverability and response to control inputs. The other condition would be a stable center of gravity that is extremely easy to control, but will not respond to abrupt control inputs or needs for high maneuverability. Placement of the control circuit on the aircraft would be important as well in an effort to reduce electrical noise from the motor. Therefore, the circuit should be placed on the fuselage somewhere near the center of gravity, which will be at enough distance to reduce the noise significantly.

The first consideration before trying autonomous flight control is autonomous flight stabilization. While in flight, one of these weights will cause the aircraft to deviate from straight and level flight. The codes would command the microprocessor to constantly monitor the real-time tracking data from the control circuit. Once a deviation is discovered to be outside the acceptable pre-programmed limits, the microprocessor would send commands to the control circuit to manipulate the

necessary flight controls until the aircraft was brought back into an acceptable configuration.

Once the flight stabilization has been achieved, the next step is automated flight. This task builds upon the previous task of flight stabilization. In addition to monitoring the aircraft's orientation in three dimensions, the microprocessor will monitor the aircraft's trajectory according to a pre-determined route. Should the aircraft deviate from the course; the microprocessor will signal the control circuit to manipulate the necessary flight controls in order to return the aircraft to the proper course. In other words, the microprocessor will monitor orientation stability, flight path stability, and also control other aspects of the flight such as holding an altitude, or maintaining a specific rate of climb or descent.

Once the previous goals are achieved, the main idea would be to adapt the aircraft and its controls to make them more user-friendly. As a physicist, a common concern is a user-friendly interface if the final product is to be used by someone else. Therefore, some aesthetics would be added to the system, such as the ability to program the flight plan through a tablet pc or even a palm pilot. This would allow the pilot to draw the flight plan in three dimensions from start to finish, then upload it to the aircraft and send it on its way.

Other considerations are the possible uses for such an aircraft. The largest possibility is military use as a completely autonomous Unmanned Aerial Vehicle. To accommodate this possibility, more considerations would be needed to allow tasks to be performed in addition to primary flight directives. For example, servos could be

mounted into the wings to control pylons holding fuel tanks, rockets, or cargo. It would be necessary for an autonomous aircraft such as this to be able to perform mission directives in addition to flying itself from point A to point B. The pilot would have an option in the pre-flight stages, after drawing up the flight plan, to add waypoints into the flight plan. At these waypoints, he or she may assign a task or function to a specific waypoint. Once airborne, the aircraft will pass the waypoint and the microprocessor will focus on performing the pre-assigned tasks before switching to the next leg of the flight path. This further accommodates any necessary military mission, from attacking a prime target to providing humanitarian assistance in a war-torn country.

Closing Statements

In one academic year, significant progress was made on this project. The groundwork was almost completed. After flight tests, this groundwork will allow the true test and construction of the prototype. The project has required the acquisition of knowledge in many crucial areas in today's career paths from structural mechanics, to signal processing, to computer programming with assembly codes. Despite the fact that not all of the goals were achieved, the crucial ground-breaking tasks were completed. Should this project be continued in the future, preparations will already be completed, and the majority of the time can be spent on the meat of the project—striving for completely autonomous flight.

Vince Lombardi once said, "Give us the tools, and we will get the work done." In this project the tools have been shaped from scratch. They have been molded and forged with the utmost of care, and are ready to be used to accomplish revolutionary feats in aviation and military technology. With this advancement, lives would no longer be lost needlessly in war. Humanitarian relief could be distributed with more efficiency and live-saving speed. All that remains is to complete the work with the tools that have been created for that very purpose. Built upon this foundation, autonomous flight can be achieved.

Appendix A-System Photographs

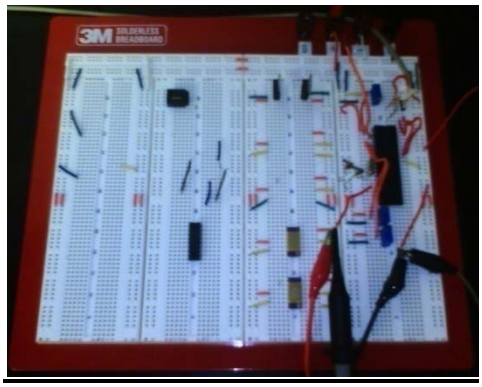

Figure A1: Microprocessor Configured for Analog-to-Digital Conversion.

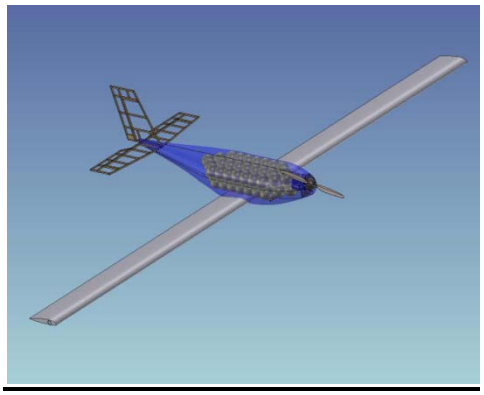

Figure A2: Simple Conventional Design Similar to the Test Aircraft [12].

Figure A3: Rate Gyros Designed for Use in a MEMS Device [11].

Figure A4: Rate Gyro Similar to One in Control Circuit [18].

Figure A5: Three-Axis Accelerometer Similar to One Used in Control Circuit [8].

Appendix B-System Diagrams

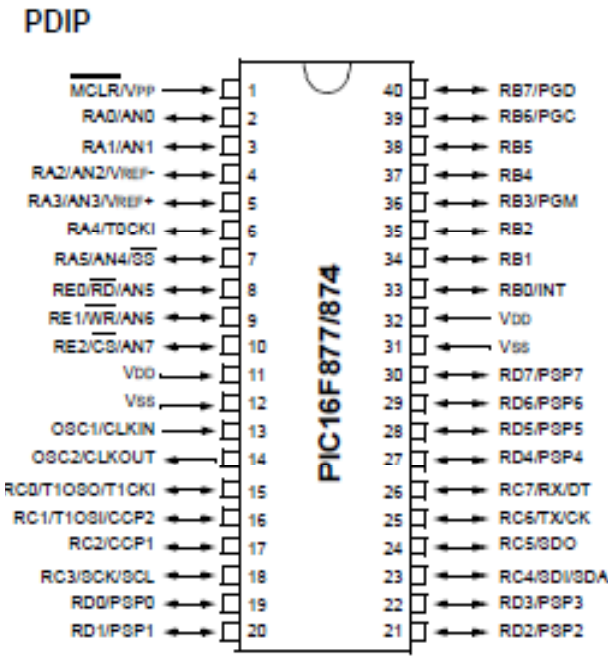

Figure B1: PIC16F877 Pin Diagram [12].

Device	Program FLASH	Data Memory	Data EEPROM
PIC16F874	4K	192 Bytes	128 Bytes
PIC16F877	8K	368 Bytes	256 Bytes

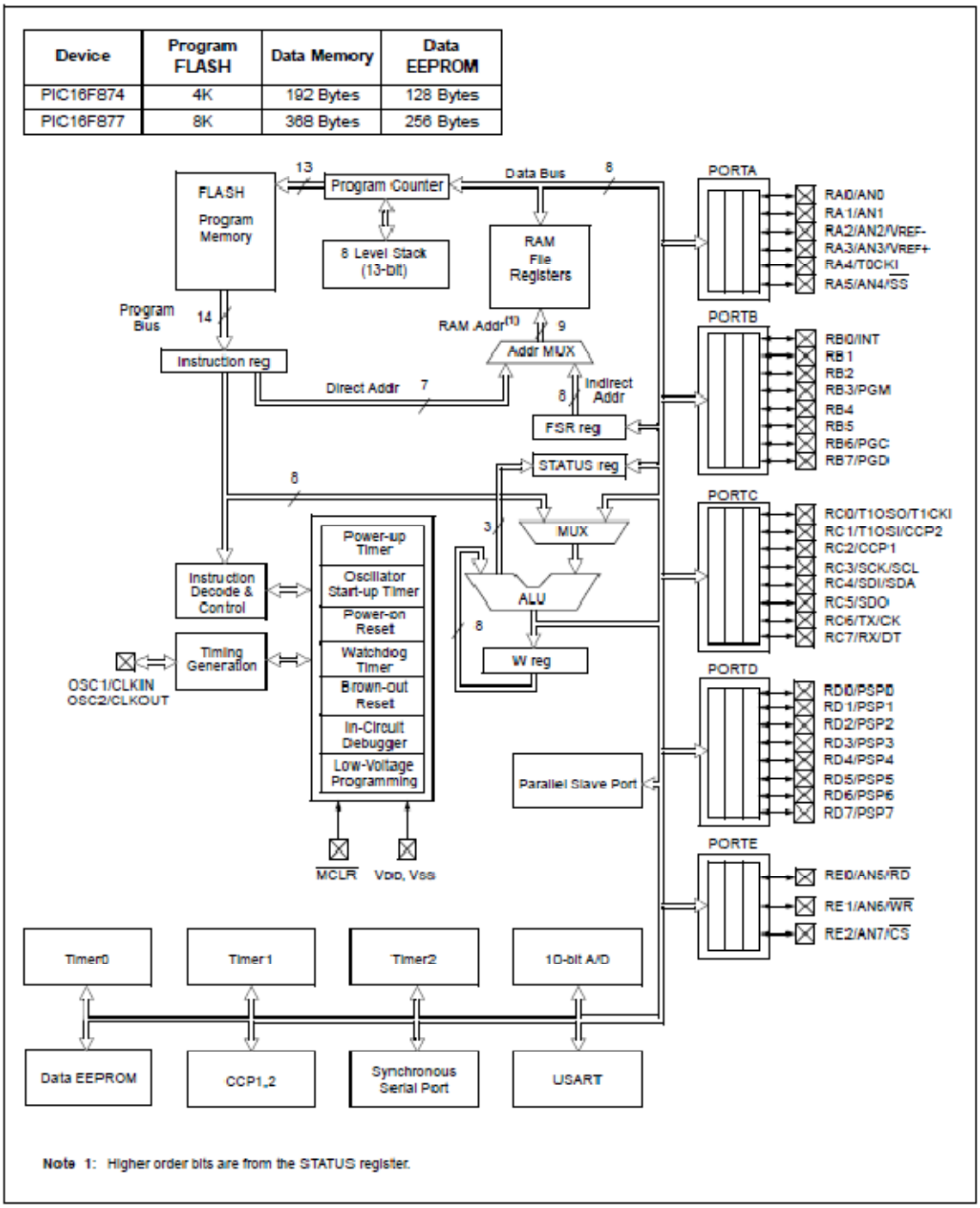

Figure B2: PIC16F877 Block Diagram [12].

Table B1: PIC16F877 Pin Descriptions [12].

Pin Name	DIP Pin#	PLCC Pin#	QFP Pin#	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	13	14	30	I	ST/CMOS(4)	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	14	15	31	O	—	Oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode. In RC mode, OSC2 pin outputs CLKOUT which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate.
$\overline{MCLR}$/VPP	1	2	18	I/P	ST	Master Clear (Reset) input or programming voltage input. This pin is an active low RESET to the device.
						PORTA is a bi-directional I/O port.
RA0/AN0	2	3	19	I/O	TTL	RA0 can also be analog input0.
RA1/AN1	3	4	20	I/O	TTL	RA1 can also be analog input1.
RA2/AN2/VREF-	4	5	21	I/O	TTL	RA2 can also be analog input2 or negative analog reference voltage.
RA3/AN3/VREF+	5	6	22	I/O	TTL	RA3 can also be analog input3 or positive analog reference voltage.
RA4/T0CKI	6	7	23	I/O	ST	RA4 can also be the clock input to the Timer0 timer/ counter. Output is open drain type.
RA5/$\overline{SS}$/AN4	7	8	24	I/O	TTL	RA5 can also be analog input4 or the slave select for the synchronous serial port.
						PORTB is a bi-directional I/O port. PORTB can be soft-ware programmed for internal weak pull-up on all inputs.
RB0/INT	33	36	8	I/O	TTL/ST(1)	RB0 can also be the external interrupt pin.
RB1	34	37	9	I/O	TTL	
RB2	35	38	10	I/O	TTL	
RB3/PGM	36	39	11	I/O	TTL	RB3 can also be the low voltage programming input.
RB4	37	41	14	I/O	TTL	Interrupt-on-change pin.
RB5	38	42	15	I/O	TTL	Interrupt-on-change pin.
RB6/PGC	39	43	16	I/O	TTL/ST(2)	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming clock.
RB7/PGD	40	44	17	I/O	TTL/ST(2)	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming data.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as an external interrupt.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.
3: This buffer is a Schmitt Trigger input when configured as general purpose I/O and a TTL input when used in the Parallel Slave Port mode (for interfacing to a microprocessor bus).
4: This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

Table B2: PIC16F877 Pin Descriptions Continued [12].

Pin Name	DIP Pin#	PLCC Pin#	QFP Pin#	I/O/P Type	Buffer Type	Description
						PORTC is a bi-directional I/O port.
RC0/T1OSO/T1CKI	15	16	32	I/O	ST	RC0 can also be the Timer1 oscillator output or a Timer1 clock input.
RC1/T1OSI/CCP2	16	18	35	I/O	ST	RC1 can also be the Timer1 oscillator input or Capture2 input/Compare2 output/PWM2 output.
RC2/CCP1	17	19	36	I/O	ST	RC2 can also be the Capture1 input/Compare1 output/PWM1 output.
RC3/SCK/SCL	18	20	37	I/O	ST	RC3 can also be the synchronous serial clock input/ output for both SPI and I^2C modes.
RC4/SDI/SDA	23	25	42	I/O	ST	RC4 can also be the SPI Data In (SPI mode) or data I/O (I^2C mode).
RC5/SDO	24	26	43	I/O	ST	RC5 can also be the SPI Data Out (SPI mode).
RC6/TX/CK	25	27	44	I/O	ST	RC6 can also be the USART Asynchronous Transmit or Synchronous Clock.
RC7/RX/DT	26	29	1	I/O	ST	RC7 can also be the USART Asynchronous Receive or Synchronous Data.
						PORTD is a bi-directional I/O port or parallel slave port when interfacing to a microprocessor bus.
RD0/PSP0	19	21	38	I/O	ST/TTL(3)	
RD1/PSP1	20	22	39	I/O	ST/TTL(3)	
RD2/PSP2	21	23	40	I/O	ST/TTL(3)	
RD3/PSP3	22	24	41	I/O	ST/TTL(3)	
RD4/PSP4	27	30	2	I/O	ST/TTL(3)	
RD5/PSP5	28	31	3	I/O	ST/TTL(3)	
RD6/PSP6	29	32	4	I/O	ST/TTL(3)	
RD7/PSP7	30	33	5	I/O	ST/TTL(3)	
						PORTE is a bi-directional I/O port.
RE0/$\overline{RD}$/AN5	8	9	25	I/O	ST/TTL(3)	RE0 can also be read control for the parallel slave port, or analog input5.
RE1/$\overline{WR}$/AN6	9	10	26	I/O	ST/TTL(3)	RE1 can also be write control for the parallel slave port, or analog input6.
RE2/$\overline{CS}$/AN7	10	11	27	I/O	ST/TTL(3)	RE2 can also be select control for the parallel slave port, or analog input7.
VSS	12,31	13,34	6,29	P	—	Ground reference for logic and I/O pins.
VDD	11,32	12,35	7,28	P	—	Positive supply for logic and I/O pins.
NC	—	1,17,28, 40	12,13, 33,34		—	These pins are not internally connected. These pins should be left unconnected.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as an external interrupt.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.
3: This buffer is a Schmitt Trigger input when configured as general purpose I/O and a TTL input when used in the Parallel Slave Port mode (for interfacing to a microprocessor bus).
4: This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	—	ADON
bit 7							bit 0

bit 7 6 **ADCS1:ADCS0**: A/D Conversion Clock Select bits
00 = FOSC/2
01 = FOSC/8
10 = FOSC/32
11 = FRC (clock derived from the internal A/D module RC oscillator)

bit 5-3 **CHS2:CHS0**: Analog Channel Select bits
000 = channel 0, (RA0/AN0)
001 = channel 1, (RA1/AN1)
010 = channel 2, (RA2/AN2)
011 = channel 3, (RA3/AN3)
100 = channel 4, (RA5/AN4)
101 = channel 5, (RE0/AN5)(1)
110 = channel 6, (RE1/AN6)(1)
111 = channel 7, (RE2/AN7)(1)

bit 2 **GO/DONE**: A/D Conversion Status bit
If ADON = 1:
1 = A/D conversion in progress (setting this bit starts the A/D conversion)
0 = A/D conversion not in progress (this bit is automatically cleared by hardware when the A/D conversion is complete)

bit 1 **Unimplemented**: Read as '0'

bit 0 **ADON**: A/D On bit
1 = A/D converter module is operating
0 = A/D converter module is shut-off and consumes no operating current

Note 1: These channels are not available on PIC16F873/876 devices.

Figure B3: Analog-to-Digital Converter Register Bit Descriptions [12].

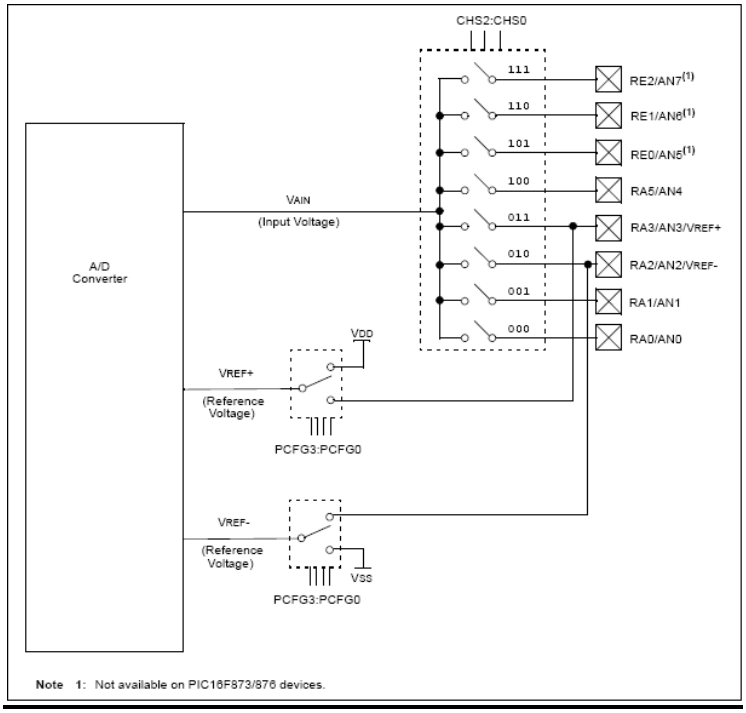

Figure B4: Analog-to-Digital Converter Block Diagram [12].

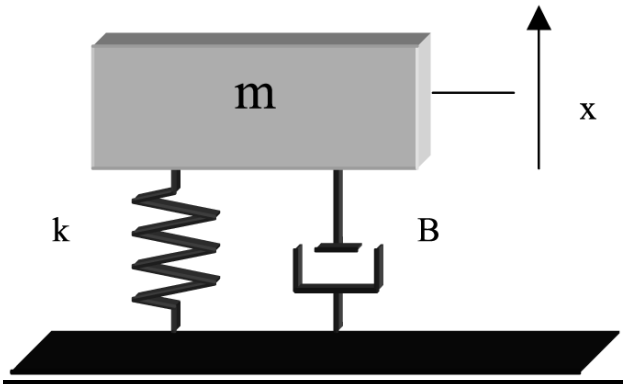

Figure B5: Simple Schematic of Accelerometer Mechanism [4].

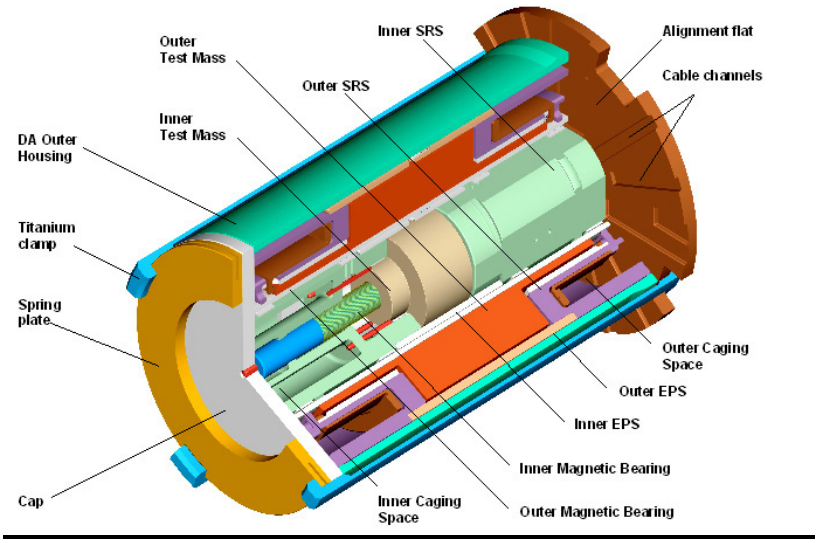

Figure B6: Cutaway of a Drum-Type Accelerometer [15].

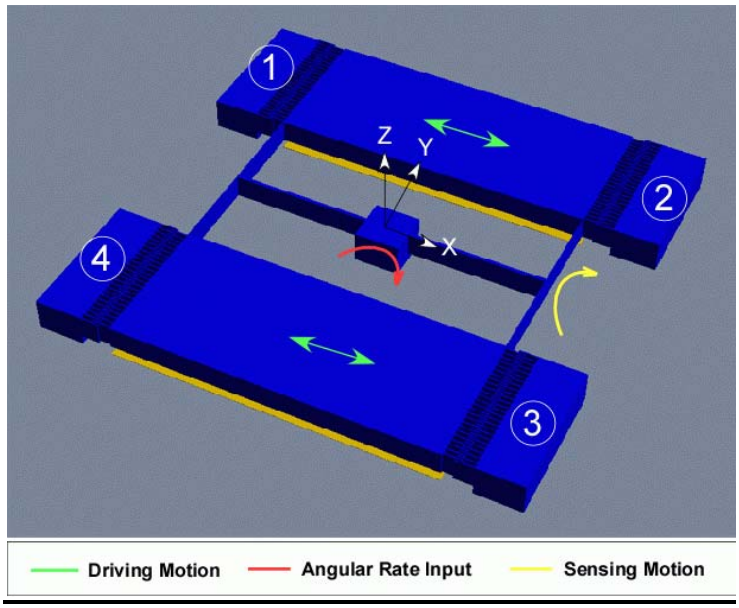

Figure B7: Rate Gyro Simulation Diagram [14].

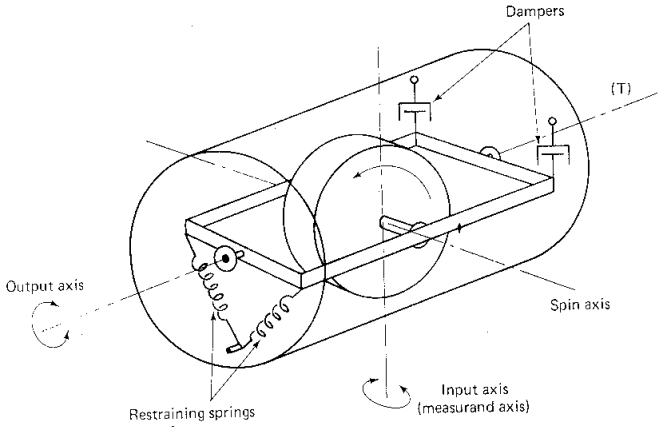

Figure B8: Rate Gyro Schematic [9].

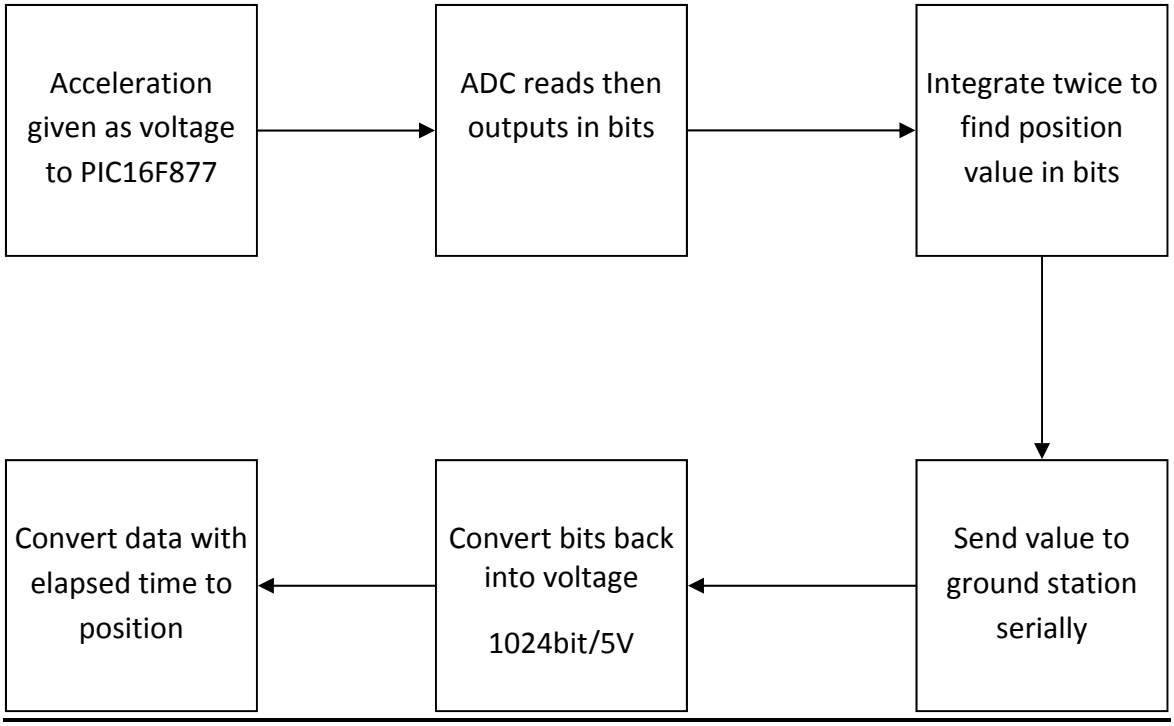

Figure B9: Acceleration Algorithm Flowchart.

Appendix C-System Codes

Code C1: Accelerometer Codes.

```
DEFINE OSC 20

'Connect x analog input to channel-0 (RA0)
'Connect y analog input to channel-1 (RA1)
'Connect z analog input to channel-2 (RA2)
'Set ADC Parameters
DEFINE ADC_BITS 10          'Set output to 10 bits
DEFINE ADC_CLOCK 3          'Set clock source
DEFINE ADC_SAMPLEUS 50      'Set sample rate 50us (min is 19.72us)
TRISA = 255                 'Set PORTA to all inputs
ADCON1 = 0                  'Set PORTA to analog and right justify results
ADCON1.7 = 1                'Last 6 digits in Binary readout are 0

'Set variable parameters
dt VAR WORD
a0x VAR WORD
a0y VAR WORD
a0z VAR WORD
a1x VAR WORD
a1y VAR WORD
a1z VAR WORD
v0x VAR WORD
v0y VAR WORD
v0z VAR WORD
v1x VAR WORD
v1y VAR WORD
v1z VAR WORD
x0 VAR WORD
y0 VAR WORD
z0 VAR WORD

'Set Debug Parameters
DEFINE DEBUG_REG PORTB      'Set debug pin port to PORTB
DEFINE DEBUG_BIT 0          'Set debug pin bit to PORTB.0
DEFINE DEBUG_BAUD 9600      'Set debug baud rate
DEFINE DEBUG_MODE 1         'Set debug mode to inverted
```

```
'Begin real time data sample and integrate for velocity
a0x = 0                 'Set initial values
a0y = 0
a0z = 0
v0x = 0
v0y = 0
v0z = 0
x0 = 0
y0 = 0
z0 = 0
dt = 1                  'Set sample time interval
                        'Note:  For scaling, 1=100ms
ADCIN 0, a1x            'Sample from pin RA0 and store as a1x
ADCIN 1, a1y
ADCIN 2, a1z
'Note:  Ave A is 472 so V<472 is interpreted as (-) by ground station.
IF a1x > 472 THEN
v0x = (((a1x+a0x)/2)*dt) - 472   'Averaging method computation of velocity
a0x = a1x
ELSE
v0x = 472 - (((a1x+a0x)/2)*dt)
a0x = a1x
ENDIF
IF a1y > 472 THEN
v0y = (((a1y+a0y)/2)*dt) - 472   'Averaging method computation of velocity
a0y = a1y
ELSE
v0y = 472 - (((a1y+a0y)/2)*dt)
a0y = a1y
ENDIF
IF a1z > 472 THEN
v0z = (((a1z+a0z)/2)*dt) - 472   'Averaging method computation of velocity
a0z = a1z
ELSE
v0z = 472 - (((a1z+a0z)/2)*dt)
a0z = a1z
ENDIF
loop:
ADCIN 0, a1x            'Sample from pin RA0 and store as a1x
ADCIN 1, a1y            'Sample from pin RA1 and store as a1y
ADCIN 2, a1z            'Sample from pin RA2 and store as a1z
```

```
'X direction
IF a1x > 472 OR a1x = 472 THEN
v1x = (v0x-472)+((((a1x+a0x)/2)*dt) - 472)   'Averaging method computation of velocity
a0x = a1x
ELSE
v1x = -v0x - (472-(((a1x+a0x)/2)*dt))
a0x = a1x
ENDIF
'Note:  Ave V is 32500 so V<32500 is interpreted as (-) by ground station.
IF v1x > 32500 THEN
x0 = (((v1x+v0x)/2)*dt) - 32500           'Averaging method computation of position
v0x = v1x
ELSE
x0 = 32500 - (((v1x+v0x)/2)*dt)
v0x = v1x
ENDIF
DEBUG "a0x =", DEC a0x, "  v1x=", DEC v1x, "  x0=", DEC x0,10,13
                                'Send a0x, v0x, and x0 values to computer
'Y direction
IF a1y > 472 OR a1y = 472 THEN
v1y = (v0y-472)+((((a1y+a0y)/2)*dt) - 472)   'Averaging method computation of velocity
a0y = a1y
ELSE
v1y = -v0y - (472-(((a1y+a0y)/2)*dt))  'Averaging method computation of velocity
a0y = a1y
ENDIF
IF v1y > 32500 THEN
y0 = (((v1y+v0y)/2)*dt) - 32500           'Averaging method computation of position
v0y = v1y
ELSE
y0 = 32500 - (((v1y+v0y)/2)*dt)
v0y = v1y
ENDIF
DEBUG "a0y =", DEC a0y, "  v1y=", DEC v1y, "  y0=", DEC y0,10,13
                                'Send a0y, v0y, and y0 values to computer
'Z direction
IF a1z > 472 OR a1z = 472 THEN
v1z = (v0z-472)+((((a1z+a0z)/2)*dt) - 472)   'Averaging method computation of velocity
a0z = a1z
ELSE
v1z = -v0z - (472-(((a1z+a0z)/2)*dt))
a0z = a1z
ENDIF
```

```
IF v1z > 32500 THEN
z0 = (((v1z+v0z)/2)*dt) - 32500          'Averaging method computation of position
v0z = v1z
ELSE
z0 = 32500 - (((v1z+v0z)/2)*dt)
v0z = v1z
ENDIF
DEBUG "a0z =", DEC a0z, "  v1z=", DEC v1z, "  z0=", DEC z0,10,13
                                'Send a0z, v0z, and z0 values to computer
GOTO loop
```

Code C2: Rate Gyro Codes.

```
DEFINE OSC 20

'Connect x analog input to channel-3 (RA3)
'Connect y analog input to channel-4 (RA4)
'Connect z analog input to channel-5 (RA5)
'Set ADC Parameters
DEFINE ADC_BITS 10          'Set output to 10 bits
DEFINE ADC_CLOCK 3          'Set clock source
DEFINE ADC_SAMPLEUS 50      'Set sample rate 50us (min is 19.72us)
TRISA = 255                 'Set PORTA to all inputs
ADCON1 = 0                  'Set PORTA to analog and right justify results
ADCON1.7 = 1                'Last 6 digits in Binary readout are 0

'Set variable parameters
dt VAR WORD
v0x VAR WORD
v0y VAR WORD
v0z VAR WORD
v1x VAR WORD
v1y VAR WORD
v1z VAR WORD
x0 VAR WORD
y0 VAR WORD
z0 VAR WORD

'Set Debug Parameters
DEFINE DEBUG_REG PORTB      'Set debug pin port to PORTB
DEFINE DEBUG_BIT 0          'Set debug pin bit to PORTB.0
DEFINE DEBUG_BAUD 9600      'Set debug baud rate
DEFINE DEBUG_MODE 1         'Set debug mode to inverted

'Begin real time data sample and integrate for velocity
v0x = 0                     'Set initial values
v0y = 0
v0z = 0
x0 = 0
y0 = 0
z0 = 0
dt = 1                      'Set sample time interval
                            'Note:  For scaling, 1=100ms
```

```
loop:
 ADCIN 3, a1x              'Sample from pin RA3 and store as v1x
 ADCIN 4, a1y              'Sample from pin RA4 and store as v1y
 ADCIN 5, a1z              'Sample from pin RA5 and store as v1z
 'X direction
  IF v1x > 472 THEN
 x0 = (((v1x+v0x)/2)*dt) - 472          'Averaging method computation of position
 v0x = v1x
 ELSE
 x0 = 472 - (((v1x+v0x)/2)*dt)
 v0x = v1x
 ENDIF
 DEBUG "v1x=", DEC v1x, "  x0=", DEC x0,10,13
                           'Send v0x, and x0 values to computer
 'Y direction
  IF v1y > 472 THEN
 y0 = (((v1y+v0y)/2)*dt) - 472          'Averaging method computation of position
 v0y = v1y
 ELSE
 y0 = 472 - (((v1y+v0y)/2)*dt)
 v0y = v1y
 ENDIF
 DEBUG "v1y=", DEC v1y, "  y0=", DEC y0,10,13
                           'Send v0y, and y0 values to computer
 'Z direction
 IF v1z > 472 THEN
 z0 = (((v1z+v0z)/2)*dt) – 472          'Averaging method computation of position
 v0z = v1z
 ELSE
 z0 = 472 - (((v1z+v0z)/2)*dt)
 v0z = v1z
 ENDIF
 DEBUG "v1z=", DEC v1z, "  z0=", DEC z0,10,13
                           'Send v0z, and z0 values to computer
 GOTO loop
```

Code C3: Serial Communication Sample Code [12].

```
' SERIN & SEROUT Commands
' Upper case serial filter.

SO          CON         0           ' Define serial out pin
SI          CON         1           ' Define serial in pin
N2400       CON         4           ' Set serial mode
B0          VAR         BYTE

loop:
  SERIN SI,N2400,B0                 ' B0 = input character
  IF (B0 < "a") or (B0 > "z") THEN print   ' If lower case, convert to upper
  B0 = B0 - $20

print:
  SEROUT SO,N2400,[B0]              ' Send character
  GOTO loop                         ' Forever
```

Code C4: Analog-to-Digital Conversion Sample Code [12].

```
' PicBasic Pro program to display result of
' 10-bit A/D conversion on LCD
' Connect analog input to channel-0 (RA0)

' Define LCD registers and bits
DEFINE LCD_DREG     PORTD
DEFINE LCD_DBIT     4
DEFINE LCD_RSREG    PORTE
DEFINE LCD_RSBIT    0
DEFINE LCD_EREG     PORTE
DEFINE LCD_EBIT     1

' Define ADCIN parameters
DEFINE ADC_BITS        10           ' Set number of bits in result
DEFINE ADC_CLOCK       3            ' Set clock source (3=rc)
DEFINE ADC_SAMPLEUS    50           ' Set sampling time in uS
adval   VAR     WORD                ' Create adval to store result

TRISA = %11111111                   ' Set PORTA to all input
ADCON1 = %10000010                  ' Set PORTA analog and right justify result
LOW PORTE.2                         ' LCD R/W line low (W)
PAUSE 500                           ' Wait .5 second

loop:
  ADCIN 0, adval                    ' Read channel 0 to adval
  LCDOUT $fe, 1                     ' Clear LCD
  LCDOUT "Value: ", DEC adval       ' Display the decimal value
  PAUSE 100                         ' Wait .1 second
  GOTO loop                         ' Do it forever
  END
```

Appendix D-Equations

$$x = \int v\,dt = \iint a\,dtdt$$

Equation D1: x = position, v = velocity, a = acceleration, and dt = constant time interval.

$$v1 = \left[\frac{(a1 - a0)}{2}\right] * dt$$

Equation D2: v = velocity, a = acceleration, and dt = constant time interval.

$$x1 = \left[\frac{(v1 - v0)}{2}\right] * dt$$

Equation D3: x = position, v = velocity, and dt = constant time interval.

$$S = 2^n + 1$$

Equation D4: S = integer bit space and n = bit capacity of the ADC (i.e. 10 bit => n=10).

Works Cited

[1] Analog Devices. High Accuracy, Dual-Axis Digital Inclinometer and Accelerometer. Norwood, 2008.

[2] Analog Devices. Wide Bandwidth Yaw Rate Gyroscope with SPI. Norwood, 2008.

[3] Columbia Research Labs, Inc. Triaxial Accelerometers. Woodlyn, 2008.

[4] Emerald. (n.d.). Retrieved April 10, 2009, from Emerald: http://www.emeraldinsight.com/fig/1740250318006.png

[5] Endevco. Piezoresistive Pressure Transducer. San Juan Capistrano, 2008.

[6] Gladiator Technologies, Inc. MEMS Landmark 10 GPS/AHRS. 2007.

[7] Heatherington, D. (n.d.). Retrieved April 10, 2009, from wa4dsy.net: http://www.wa4dsy.net/robot/uploads/images/balance-bot/DSCF0753.jpg

[8] HVW Technologies. (2009). Retrieved April 11, 2009, from hvwtech.com: http://www.hvwtech.com/products/542/35220_PV.jpg.

[9] Istanbul Technical University. (2009). Retrieved April 11, 2009, from gidb.itu.edu.tr: http://www.gidb.itu.edu.tr/staff/unsan/D/yleb/experimental/not/sensor/l17/rate_gyro.gif

[10] MEMSense. Bluetooth Inertial Measurement Unit Series Documentation. Rapid City, 2008.

[11] MEMsTech. Uncompensated, Absolute Pressure Sensor. Northville, 2008.

[12] microEngineering Labs. PIC16F87X Data Sheet. 2001.

[13] microEngineering Labs. PICBasic Pro Compiler. Colorado Springs: microEngineering Labs, 2004.

[14] Solidus Technologies, Inc. (2008). Retrieved April 10, 2009, from Solidus Technologies, Inc.: http://www.solidustech.com/images/MEMS_ElementSimulationDiagram.jpg

[15] Stanford University. (2008). *Gravity Probe B: Testing Einstein's Universe*. Retrieved April 11, 2009, from Einstein.stanford.edu: http://einstein.stanford.edu/STEP/information/data/accelerometer_2.jpg

[16] Tomayk, James E. Computers Take Flight: A History of NASA's Pioneering Digital Fly-By-Wire Project. Washington D.C.: NASA History Office, 2000.

[17] Watson Industries, Inc. Fluxgate Magnetometer. Eau Claire, 2008.

[18] Watson Industries, Inc. (2008). Retrieved April 10, 2009, from Watson-gyro.com: http://www.watson-gyro.com/images/VSG_product.jpg.

[19] Witte, P. (2009). Retrieved April 10, 2009, from Witte Aero: http://witte-aero.com/Images/dbf/fuse6.JPG.