SENSOR INTEGRATION FOR IMPLEMENTATION OF OBSTACLE AVOIDANCE IN AN AUTONOMOUS HELICOPTER SYSTEM

A Thesis

by

CHRISTOPHER ISAAC MENTZER

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2006

Major Subject: Mechanical Engineering

SENSOR INTEGRATION FOR IMPLEMENTATION OF OBSTACLE AVOIDANCE IN AN AUTONOMOUS HELICOPTER SYSTEM

A Thesis

by

CHRISTOPHER ISAAC MENTZER

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee, Reza Langari
Committee Members, Darbha Swaroop
John Hurtado
Head of Department, Dennis O'Neal

May 2006

Major Subject: Mechanical Engineering

ABSTRACT

Sensor Integration for Implementation of Obstacle Avoidance in an Autonomous Helicopter System. (May 2006)

Christoper Isaac Mentzer, B.S., Texas A&M University

Chair of Advisory Committee: Dr. Reza Langari

This thesis describes the development of the Texas A&M University Autonomous Helicopter System and the integration of obstacle avoidance capabilities into that system. The helicopter system, composed of a Bergen Observer helicopter and a Rotomotion Autonomous Flight Control System (AFCS), was developed as a platform to support the development of the obstacle avoidance system through integration of sensors and onboard processing capabilities. The system has proven in various flight tests that it has the capability to autonomously hover and fly to user defined GPS waypoints. The obstacle avoidance algorithm has been proven in simulations involving an interface with the Rotomotion AFCS and the flight simulation software they created to facilitate the development of that system. The helicopter has also demonstrated appropriate responses to sensor input commensurate with the obstacle avoidance algorithm. Full avoidance tests were unable to be performed due to hardware malfunctions inherent in the obstacle avoidance sensors.

DEDICATION

"For the foolishness of God is wiser than man's wisdom, and the weakness of God is stronger than man's strength … so that your faith might not rest on men's wisdom, but on God's power." (1 Corinthians 1:25, 2:5)

All glory to God for the opportunity to start this and the gifts to finish it.

Unending gratitude to my family for their love, support and encouragement through this time and throughout my life.

ACKNOWLEDGMENTS

- Thesis advisor
  - Dr. Reza Langari
- Director – Texas A&M AVSI
  - Mr. David Lund
- Pilots
  - Mr. Anthony Jager
  - Mr. Curtis Youngblood
- Rotomotion
  - Mr. Dennis D’Annunzio
  - Mr. John Bornish
- Intec Automation
  - Mr. Mike Lavender
- Masters student
  - Mr. Joshua Davis

TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

Interest in unmanned aerial vehicles (UAVs) has grown in recent years due to the many benefits they provide over manned vehicles. First and foremost unmanned vehicles eliminate the possibility of loss of life if the aircraft were ever to crash. Secondly, since the aircraft does not have to carry a person or people, it can be much smaller and possibly perform more tasks in harsher environments. They can also be used to perform long term tasks that may be tedious or difficult for a human to do for extended periods of time. The Department of Defense describes these needs as the "dull, dirty and dangerous" missions and provides them as the main justification for research on UAVs [10]. In addition, helicopter UAVs provide the capability of flying at low altitude and hovering in one place.

The Defense Advanced Research Projects Agency (DARPA), which allocates funds for the government for defense research, has funded research projects for both unmanned airplanes and helicopters. Many universities and private companies have also ventured into the UAV helicopter industry. Major names in the aeronautics industry such as Northrop Grumman, Lockheed and Boeing have developed unmanned helicopter systems stemming from DARPA research funds [3, 5]. In their Unmanned Aircraft Systems Roadmap 2005-2030, the Department of Defense (DoD) has outlined the current state of UAVs in the world and the proposed direction of research in this area for the next several years. The major rotorcraft system listed in this report is the Northrop Grumman RQ-8A/B Fire Scout. Future rotorcraft plans include the Boeing/Frontier A160

This thesis follows the style of the *IEEE Journal on Selected Areas in Communications*.

Hummingbird, the DARPA DP-5X, and the Bell Textron Eagle Eye tilt rotor aircraft. The UGV-UA (unmanned ground vehicle – unmanned aircraft) Cooperative Development at SPAWAR Systems Center San Diego program listed in DoD report lists a Rotomotion helicopter, which uses the same flight control system as the one used in this thesis, as a possibility for the aerial vehicle system [10].

The International Aerial Robotics Competition held by the Association for Unmanned Vehicle Systems International (AUVSI) has been held annually with universities such as the Georgia Institute of Technology (Georgia Tech), the University of Arizona, and the University of Texas at Austin competing. Four universities have completed the Level 1 requirements of the test which requires the craft to fly over a distance of 3 km and hover at a designated position at the end of that flight. Only the Georgia Institute of Technology has completed Level 2 of the competition which requires that the craft implement sensors to identify to target building and at least one opening in that building through which it can enter [1, 2]. The Georgia Tech system involves an integration of a helicopter, ground vehicle and ducted fan vehicle as an approach to accomplishing all of the requirements set forth by the AUVSI for their competition [10].

This paper discusses the development of an autonomous helicopter system at Texas A&M University. First the integration and development of the hardware system as a platform for experimentation is discussed. The Rotomotion flight control software and obstacle avoidance software are then explained and discussed. Finally the experimental flight results are discussed and recommendations are made for considerations in future research.

HARDWARE

Mechanical

Figure 1: Bergen Observer

The Texas A&M University autonomous helicopter system is built around a Bergen Observer model helicopter (see Figure 1). When assembled with the original factory parts this helicopter is 52" long, 17 1/2" tall, and weighs approximately 16 pounds. It has 800 mm carbon fiber V-Blades on the main rotor and 95 mm tail rotor blades. The older models of this craft (such as the one used in this experiment) come with a Zenoah G-23 engine while the newer versions now come with the G-26 engine from Zenoah that provides an additional 800 rpm to the range of its predecessor. With

the factory configuration, the Observer is capable of carrying an 8 lb payload. Bergen is well known for developing helicopters that cater to industrial applications. The Observer is no exception in that it has a camera platform mounted on the front, which can be controlled separately. The camera mount provides a 270º field of view with capabilities for both yaw and pitch rotations

The Texas A&M Bergen Observer was donated by the Air Force in a partially assembled state. Upon receipt of the craft, the missing parts were ordered. Subsequently Mr. Anthony Jager, a 10 year veteran R/C pilot, was enlisted on a volunteer basis to check for flight preparedness and to be our test pilot. Mr. Jager works closely with Mr. Curtis Younblood, a world champion R/C helicopter pilot who also helped fly the Bergen Observer late in the project. Mr. Jager looked over our helicopter and provided his mechanical expertise to make the craft flight ready. Upon his recommendation, we ordered heavier V-Blades (still 800mm but weighing approximately 200g) to help increase the stability of the helicopter. We also purchased 110mm carbon fiber tail blades to enhance the stability of the vehicle. Concerned about the sluggish response of the standard servo operating the collective, we also replaced that servo with a Futaba model DS8311 high torque servo. After initial test flights, concerns arose about the payload capacity of the helicopter As a result the stock G-23 engine was swapped out for the more powerful, larger bore G-26 engine. This was an easy swap in that both engines have the same general outer dimensions allowing one to unbolt the old engine from the frame and bolt the new engine in its place without having to modify the mounting area or change the gear ratios of the helicopter.

A final modification that was necessary for the Observer arose from fact that the tail rotor is driven from the main engine shaft by a nylon belt that is rotating through the hollow aluminum tail boom at a high rate of speed. This motion works under the principle similar to that of a Van de Graaff generator and creates high levels of static electricity. This in conjunction with the non-conducting G10 body material of the helicopter created high electrical potential differences across the craft that could in turn potentially harm onboard electronics or create communication failures. This phenomenon was revealed to us by Rotomotion as they had had a lot of experience with the Bergen Observer during the development of their autonomous helicopter flight control system. It was recommended by John Dornish of Rotomotion that we ground the major components of the helicopter together in order to counteract this problem. To do this we created a conductive contact point on the tail boom of the helicopter by grinding away some of the black annodization on its underside

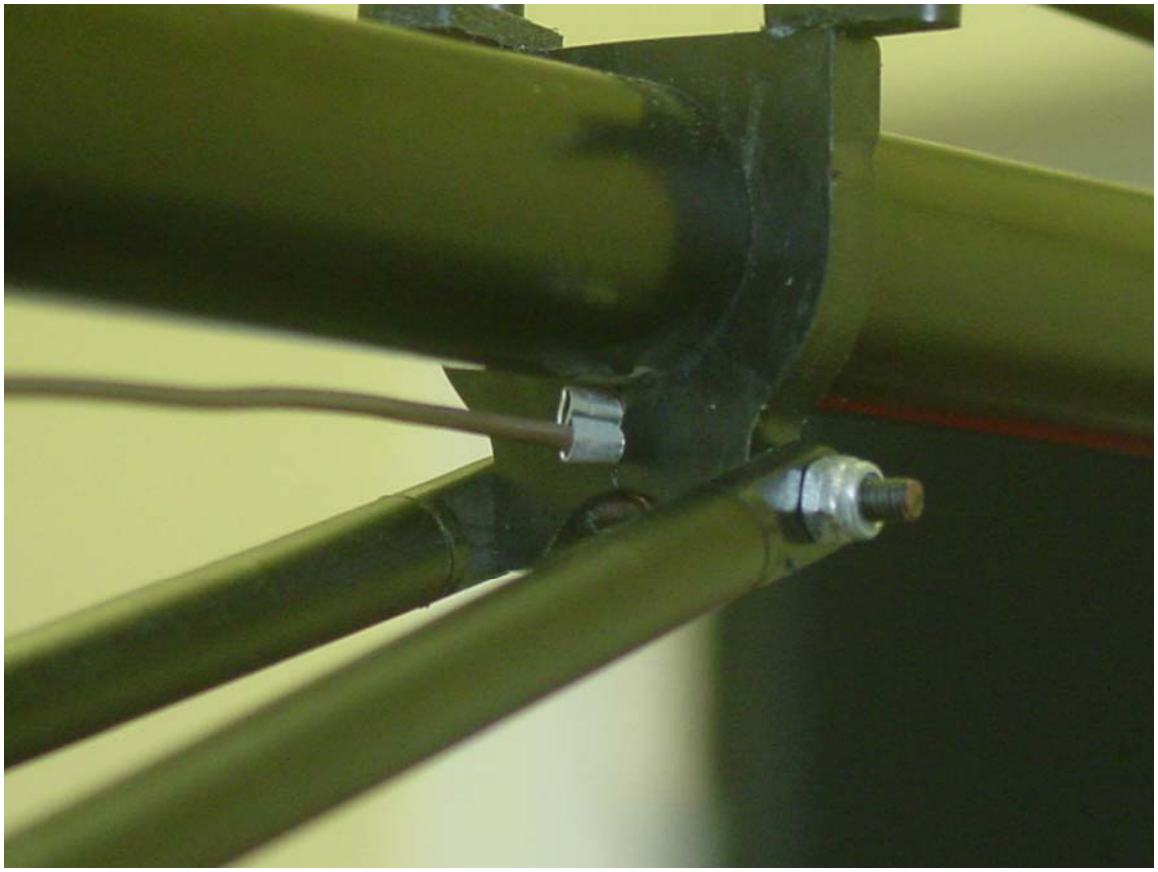

Figure 2: Tail Boom Ground

at the point where the tail boom supports come in contact with the tail boom (see Figure 2). The clamp from the tail boom supports was then used to hold an electrical terminal at the end of a wire in contact with the exposed portion of the tail boom. The other end of the wire was then affixed to the triple bearing block, which, as the name implies, is the block containing three bearings that is located directly above the helicopter engine in between the two halves of the main body frame as seen in Figure 3. Another

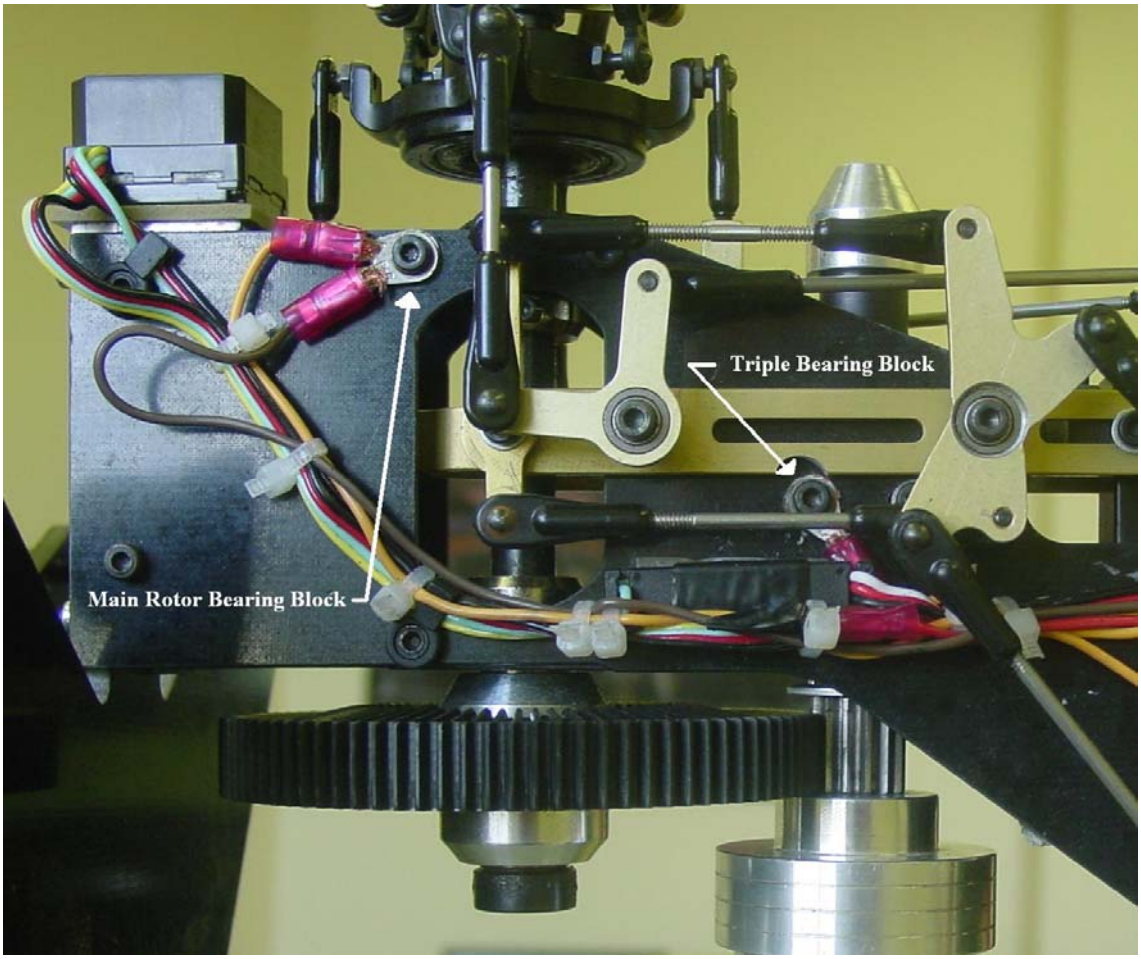

Figure 3: Grounding Points

wire was then run from that connection on the triple bearing block to the main rotor bearing block, which is also located between the two halves of the main body frame and

is used to support the main rotor. With these three points on the helicopter connected and tested for electrical conductivity, the main static generating portions of the craft were no longer isolated eliminating potential differences by use of a common ground. As a side note, the newer versions of the Bergen Observer no longer use a nylon belt driven tail rotor, but rather incorporate a rotating shaft and gear system that passes through the tail boom that could greatly reduce the static electricity problems seen in this version of the Observer.

Electrical

The electrical system for the Texas A&M University autonomous helicopter system is composed of two main components: the helicopter electronics and the ground electronics. The helicopter electronics include a Rotomotion Autonomous Flight Controller System, an Intec Automation Wildfire microcontroller, a Rotomotion modified Futaba R149DP 9 channel receiver, a regular Futaba R149DP 9 channel receiver, a safety relay board, seven Futaba servos, a Futaba GY401 gyro, a Futaba GV-1 governor, five SensComp MINI-AE ultrasonic sensors, a Dynex DX-ESW5 Ethernet router and a Renasis SAP30b wireless access point. The AFCS, Wildfire, router, and wireless access point are all powered by a single 12V battery which eliminates the need for other additional batteries that might operate at other voltages, saving precious weight on the vehicle. The regular Futaba 9 channel receiver has a 4.8V battery pack plugged into it that provides a redundant power supply that can be used to land or crash the helicopter if the AFCS loses control or the 12V battery dies. A wiring diagram of the helicopter

electrical system can be seen in Figure 4.

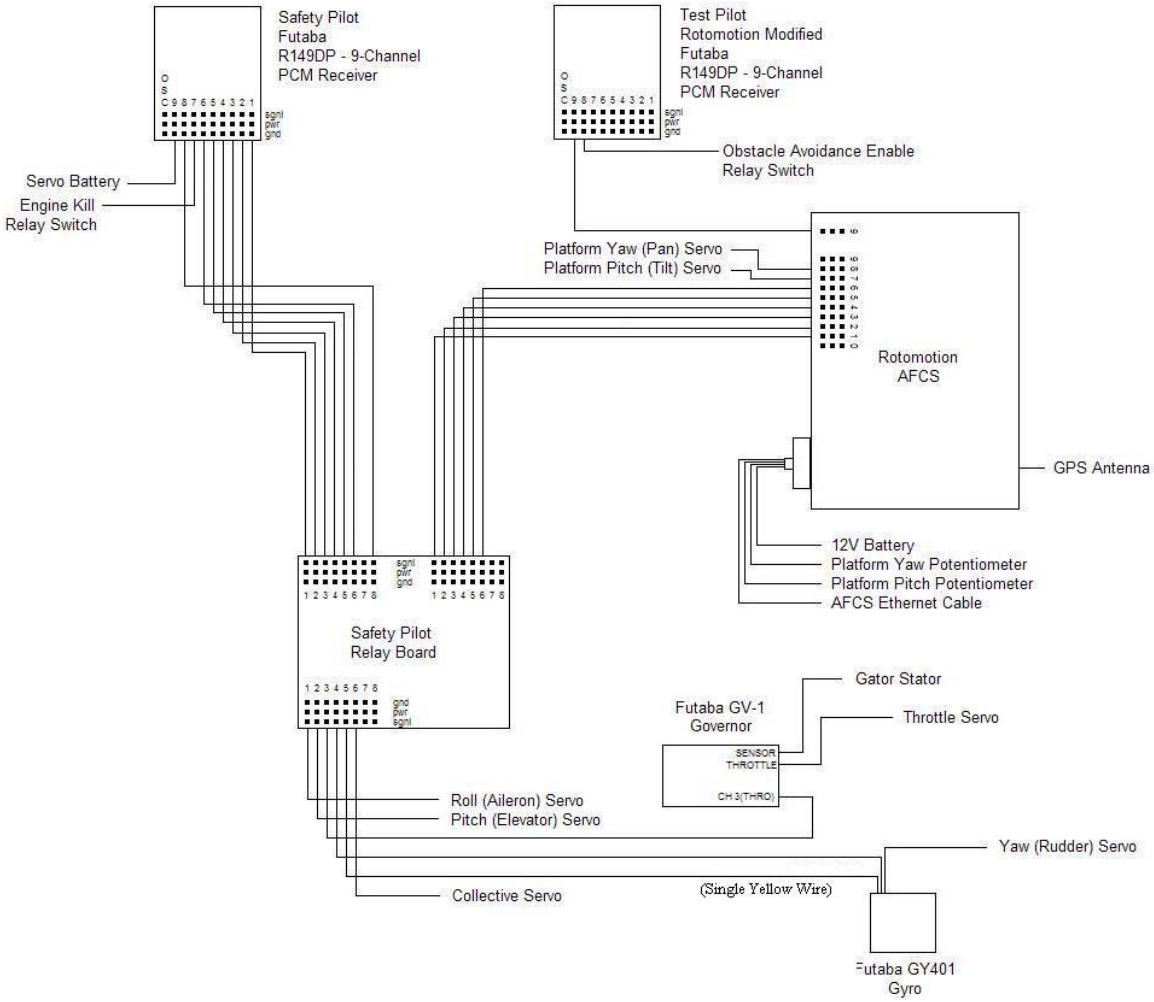

Figure 4: Helicopter Wiring Diagram

The ground electronics are a Dell Inspiron 9300 laptop, another Renasis SAP30b wireless access point, and two Futaba T9CHP 9 channel transmitters.

Helicopter Electronics

The heart of the helicopter electrical system is the Rotomotion Autonomous Flight Controller System (AFCS). This system incorporates an Altitude Heading and

Reference System (AHRS), a GPS receiver, and a servo driver that are all controlled by a local xScale processor (see Figure 5).

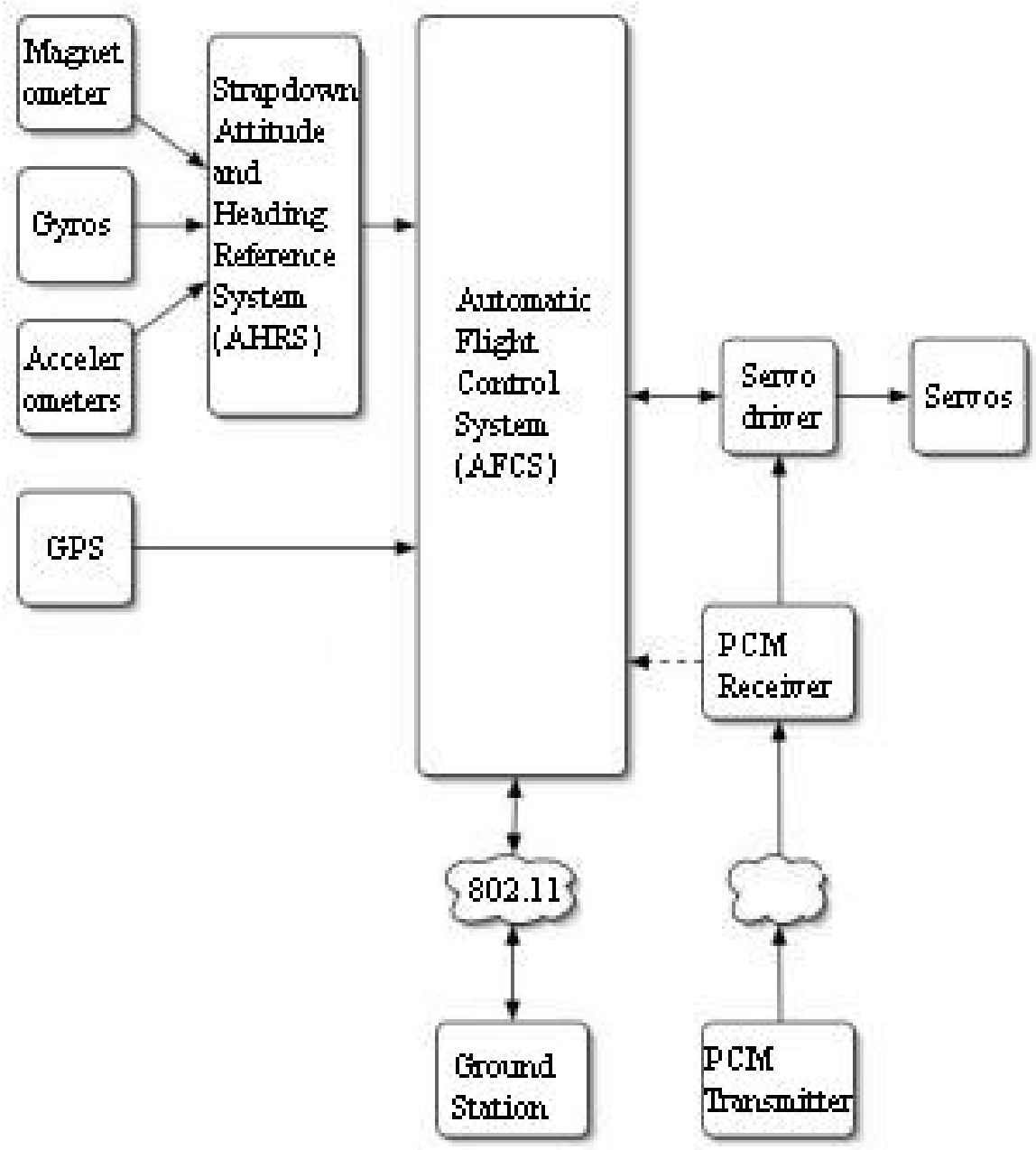

Figure 5: AFCS Configuration [11]

The AHRS system is composed of the three accelerometers to measure movement along the three linear degrees of freedom axis, three gyros to measure rotation along the three rotational degrees of freedom axis, and three magnetometers to measure relative magnetic fields and act as a compass to find both north and the plane of the ground below the helicopter. The information gathered from the AHRS is then combined with GPS data and processed through a Kalman filter to provide the flight control program with state data that can be used for control purposes. Control of the helicopter is based on several proportional-integral-derivative (PID) loops that govern the position, velocity, and attitude of the helicopter. The controller utilizes these PID loops in conjunction with the state data to keep the helicopter in a stable hover and then fly to various positions that may be defined by displacement relative to the current position of the craft in either the xyz coordinate system relative to the body frame or the north-east-down (ned) absolute coordinate system, or displacement in absolute coordinates based on the ned coordinate system or latitude and longitude coordinates. Velocity of the craft may also be controlled by either defining the velocity in m/s directly or defining a transit time between the current position and the desired position. The AFCS translates the request of the operator into helicopter movement by outputting signals to the roll, pitch, collective, throttle (via the governor) and rudder (in conjunction with the gyro) servos as well as the two servos used to the control the yaw and pitch of the camera platform. The AFCS also communicates with the ground station via its Ethernet link to receive new user commands and to inform the user of the state of the helicopter via the ground station [11].

The AFCS receives control input signals from the test pilot transmitter via the Rotomotion modified Futaba 9-channel receiver. This receiver is identical in appearance

to the other Futaba 9-channel receiver, but has been modified to output the pulse code modulated (PCM) signals from the receiver for each individual channel as one signal through the ninth channel output so that it may be connected to the AFCS through a single wire. This signal represents commands being sent from the test pilot transmitter to fly the helicopter manually and to put the helicopter into autonomous mode. The other channel outputs of the modified receiver do still output signals that can be directly connected to their respective servos as well, although this is not necessary in this case because the PCM signal sent to the AFCS is parsed by the onboard processor and implemented via the servo controller built into the AFCS. This extra processing of the signal from the modified receiver can lead to a less smooth response in the servos to pilot commands than if the servos were being controlled directly through the unmodified receiver. This servo jitter can be seen most prevalently in the rudder servo, which is presumably due to the fact that this is a quicker response time servo than the others on the craft and the fact that the signal going to this servo also has to pass through the gyro after being sent from the servo controller on the AFCS.

The safety relay board provides a redundant backup pilot system to the test pilot/Rotomotion avenue of control. This board was developed by and purchased from the Air Force. It has two input sections of pins with eight standard R/C servo three-pin connectors and one output section, also with eight three-pin connectors. The board switches between control from the safety pilot and the test pilot via the eighth three-pin connector in the safety receiver inputs (see Figure 6).

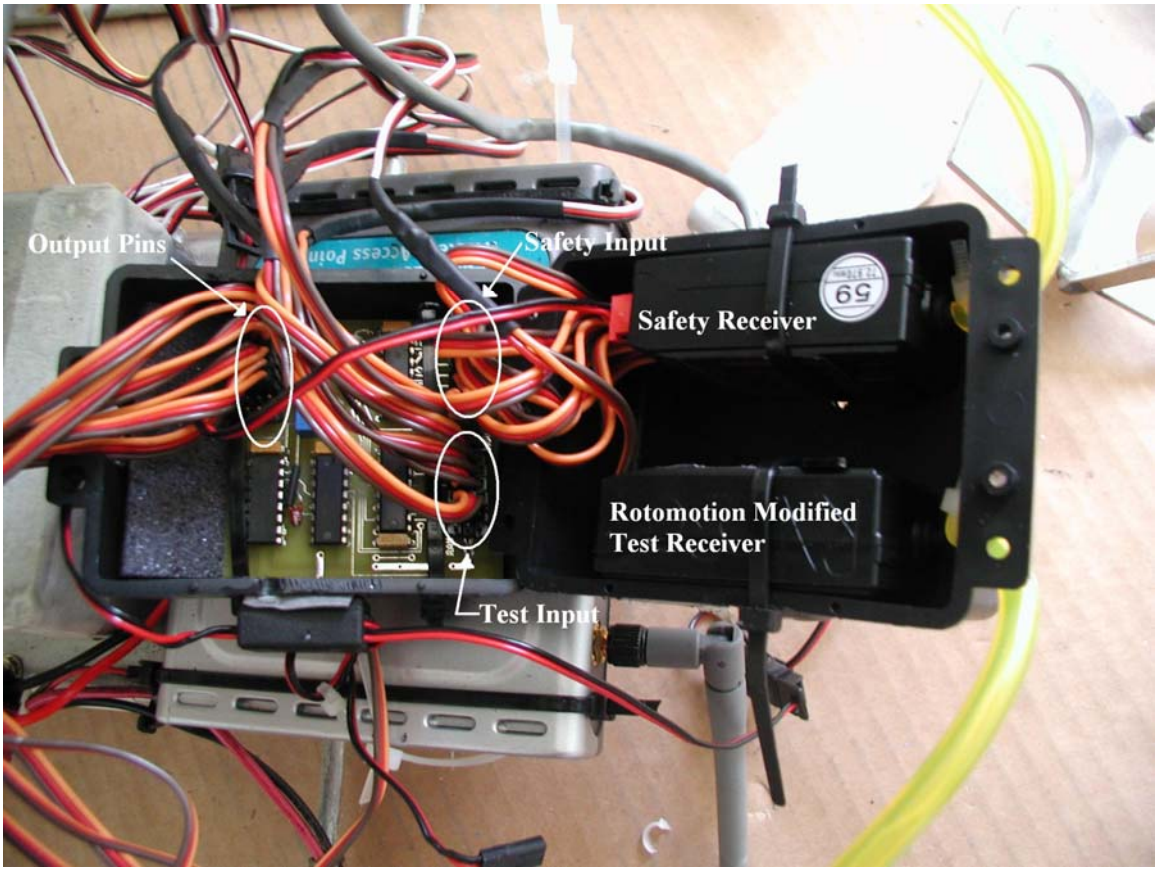

Figure 6: Safety Relay Board Wiring

The outputs from the safety receiver and the AFCS servo controller are plugged into the left and right set of input pins, respectively, in the same channel order starting with channel one to the left and continuing up through channel seven. The eighth

channel from the safety receiver is then plugged into the eighth three-pin connector of the seven safety controller channels as the safety handover switch is assigned to channel eight on the safety transmitter. The output connectors of the safety relay board are to be connected to their respective servos, with the channels again starting at one on the left and continuing upward to seven as one progresses to the right through the connectors. One key difference between the input connectors on the top of the board and the output connectors on the bottom of the board lies in the fact that the common ground on the input connectors is the bottom of the three pins, making the top pin the signal pin. In contrast, on the output section of pins the common ground is the top of the three pins, making the bottom of the three pins the signal connection. This board transfers the commands from the safety side of the inputs to the outputs on the board and ignores the test pilot side as long as no signal is sent to the switching connector. Once an ON signal is sent to the switching connector (in this case by flipping switch F of the safety transmitter which causes an ON signal to be sent through channel eight of the safety receiver) the board ignores the signals from the safety receiver (with the exception of the eighth channel that is going through the switching connector) and throughputs the signal from the test pilot input side of the board to the servos. For most cases of flight testing, control of the board should be switched over to the test pilot inputs, but if an incident occurs where the AFCS loses control and the test pilot cannot regain manual control of the craft, then control can be taken back by the safety pilot to try to recover control of the helicopter or to crash it if necessary to avoid causing damage to onlookers or property. The safety relay board also relies on power from the 4.8V battery that is plugged directly into the safety receiver so that if low battery power from the 12V battery used to power

the AFCS or hardware failure within the AFCS box is to blame for loss of control of the helicopter then the relay board and safety pilot receiver will still be able to operate independently.

The Intec Automation Wildfire microcontroller is the heart of the obstacle avoidance portion of this helicopter system. It incorporates a 64MHz FreeScale Integrated ColdFire Version 2 Microcontroller with an RJ45 socket Ethernet port and 8 analog to digital (A/D) ports among other features. This particular controller was selected because it is a small, lightweight, low-cost, low-power consumption device that has the proper processor and input/output capabilities to work with both the selected sensors and AFCS device. The function of the microcontroller is to interpret analog voltages received from the SensComp ultrasonic sensors into relative distances from the helicopter, process that information, read state data from the AFCS, derive a course of action in the event that an obstacle is detected, and implement that plan by sending commands via UDP packets through the Ethernet connection to the AFCS.

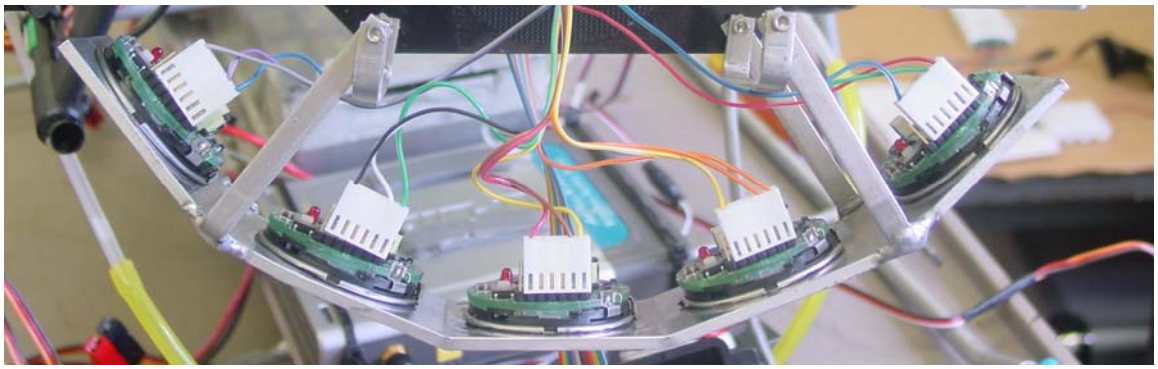

Figure 7: Sensor Array

The five SensComp Mini-AE ultrasonic sensors were selected for the obstacle avoidance system because of their relatively low cost, small size, light weight, relatively narrow

beam pattern and relatively long sensing range when compared to other potential distance sensors. Each sensor weighs 0.6 grams, is only 1.700 inches in diameter, is 0.950 inches thick, and has a usable range from one foot to 40 feet. The beam pattern emitted from the sensor is also only 15º nominal compared 30-40º for other ultrasonic sensors on the market. The sensors also have a sampling rate of 10 Hz, which is faster than the other ultrasonic sensors on the market. The environmental grade of the sensors, which are the ones used in this system, are made of 304 Stainless Steel which helps them to resist corrosion due to harsh and humid environments (SensComp). The sensors have been arranged such that there is a center sensor that is to face in the direction of the velocity of the craft and two sensors to either side of the sensor fanned out at an angle of 25º each (see Figure 7). This was done so that the respective beam patterns from each sensor would overlap, allowing for more resolution in obstacle sensing by taking into account combined (overlapped) and individualized signals (see Figure 8). This is especially important to the center sensor in that if it detects an object and the sensors on either side of it do not, then that signifies an obstacle (which if detected at 40 feet would have to be less than 14 feet in diameter) directly in the flight path of the helicopter that it can avoid by a small movement to either side of that obstacle without having to reduce speed.

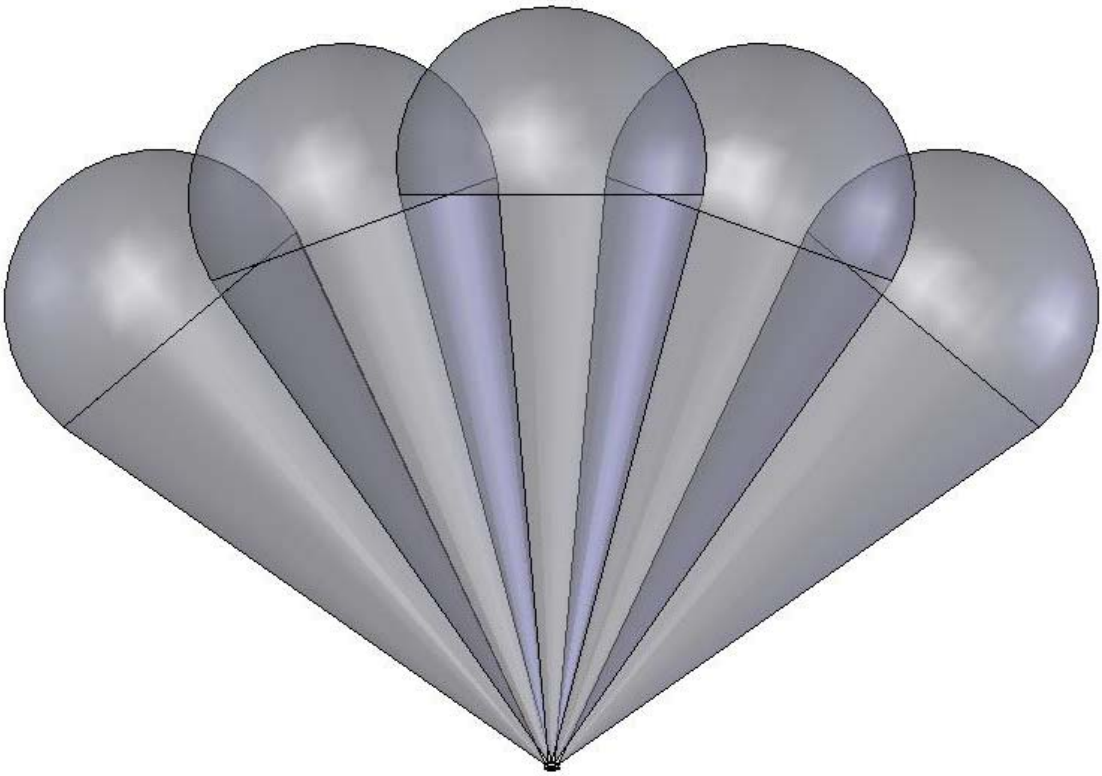

Figure 8: Sensor Beam Pattern Overlap

The SAP30b wireless access point by ITMM was chosen for both the helicopter and ground station means of data communication because it is one of the highest power wireless access points available. While most commonly used commercial access points output a signal using a little over 30 mW of RF transmit power, the SAP30b has 500 mW of output power. This level of output power in conjunction with a 9dB rubber duck style antenna provides our data communication signal with the maximum power allowed by the FCC for omni-directional communication. By utilizing the maximum power in the transmitter and minimizing the gain in the antenna, this minimizes the "pancake" effect of high gain antennas whereas when the signal is able to spread out further, it also flattens out. This reduces the signal coverage to a plane level with and perpendicular to the

antenna. A lower gain antenna has less of this “pancake” effect, thus it is more suitable for our application in which the helicopter may not be flying in just one plane, but may change altitude and will generally be flying at a height above the ground greater than that at which the antenna is located. The use of a high power transmitter also eliminates the need for an external amplifier to boost the signal, which can cause excessive noise and reduce signal clarity. The access point operates under the 802.11b protocol as opposed to a faster protocol such as the newer 802.11g protocol, which is useful because the higher bandwidth is not needed for this application and a slower protocol allows for less potential for data loss due to interruptions. Another major benefit of this wireless access point when compared to most mainstream manufacturers is that it operates off of a 12V DC input voltage, which is the same as the input voltage required for the AFCS, allowing it to use the same battery and eliminating the need for another battery that operates at a different voltage. The ability of this access point to utilize the battery pack already on board the helicopter allows us to save the space and more importantly the weight that another battery pack requires.

The Dynex DX-ESW5 router used in this helicopter system was selected because of its low cost, low weight, and most importantly its required input voltage of 12V. This model is the generic brand of Best Buy and is the only basic four port wired router that utilizes the 12V input voltage that would allow it to share a battery with the AFCS, wireless access point, and Wildfire microcontroller. This router connects the Wildfire microcontroller and the AFCS to each other and to the wireless access point to create the onboard network on the helicopter (see Figure 9). This allows the two controllers to communicate with each other and to receive and send signals to the ground station.

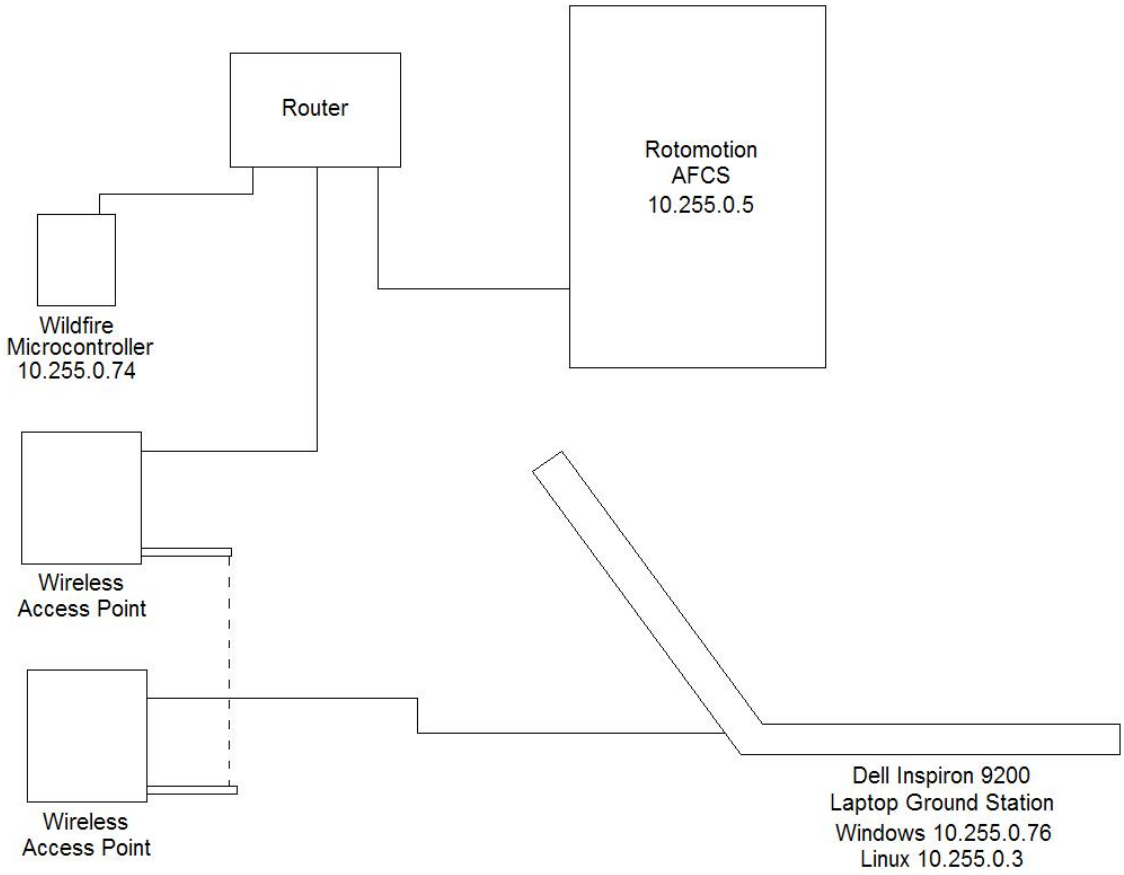

Figure 9: Helicopter System Network

Ground Electronics

The Dell Inspiron 9300 laptop computer runs the ground station software for this system. Connected to another ITMM wireless access point, the ground station receives state information from the AFCS and sends fly to and look at commands to the AFCS. The laptop can also display commands sent to the AFCS from the Wildfire microcontroller to monitor the sensor readings and reactions created by the obstacle avoidance system. The two Futaba 9 channel transmitters are for use by the test pilot and safety pilot. The channel assignments for these transmitters and for the AFCS are listed in Table 1.

Table 1: Channel Assignments

	Safety Transmitter		**Test Transmitter**		**Rotomotion AFCS**
Channel	Assignment	Channel	Assignment	Channel	Assignment
				0	Servo Battery (not used)
1	Roll (Aileron)	1	Roll (Aileron)	1	Roll
2	Pitch (Elevator)	2	Pitch (Elevator)	2	Pitch
3	Throttle	3	Throttle	3	Throttle
4	Yaw (Rudder)	4	Yaw (Rudder)	4	Yaw
5	Gyro	5	Gyro (Yaw Gain)	5	Yaw Gain
6	Collective	6	Collective	6	Collective
7	Engine Kill	7	(3-way) Autopilot Control Mode	7	Pan
8	Safety Takeover	8	Obstacle Avoidance Enable	8	Tilt
9	Servo Battery	9	(2-way) Autopilot Engage	9	not used
				9(P)	PCM Signal

SOFTWARE

Rotomotion Software

The ground station software provided by Rotomotion utilizes the state data sent by the AFCS via UDP packets over the wireless network connection to provide the operator with information as to the current status of the helicopter as well as allowing the operator to send commands to the helicopter. A screen capture of the ground station with the moving map displayed can be seen in Figure 10. An artificial horizon is also provided

Figure 10: GCS Moving Map Screen

on the ground station to show the actual yaw, pitch, and roll of the craft as well as the desired yaw, pitch, and roll if an action is requested. The area to the left of the ground station screen provides status bars pertaining to the state of the helicopter as defined by different variables. Under the map tab, the ground station provides a moving map showing the helicopters location in the absolute ned coordinate system. The user may double click on points on this moving map to either tell the helicopter where to go or where to look with its camera platform depending on whether the fly to or look at mode is selected. The information under the status tab shows the angle, position, velocity, and acceleration readings taken by the sensors on board the AFCS. The key pieces of information that can be drawn from this tab are the number of satellites the GPS has connected with and the quality of that connection. In order for the AFCS to make accurate calculations as to the position and velocity of the helicopter, it must have a good connection with a number of satellites and a good quality of signal. Generally the greater the number of satellites the GPS can see and the lower the vdop number, the better the GPS can calculate position. Also, if a WAAS satellite has been located (a satellite with an id number greater than 100) then the GPS resolution in increased even more to an accuracy of less than one meter. The number of satellites in use and the vdop rate can also be seen at the bottom of the ground station. For great satellite connectivity it is best to have more than seven satellites in the calculation with a vdop rate less than twenty. The AFCS will not even provide velocity calculations until it has made contact with at least a few satellites. The config tab and the mag tab of the ground station are used to modify the configuration parameters of the AFCS that define the PID control loops and the magnetometer relationships to define the north and down directions for the helicopter.

The logs tab shows line graphs of data being received from the AFCS, and the camera tab can be used to move the camera platform on the helicopter. More detailed information about the ground station can be found in Rotomotion's documentation.

There are two options for running the Rotomotion software on any laptop. It can either be run in Windows or in Linux. To run the ground station in Windows, one must locate the directory in which the win32 version of the executables are located (in the instance of the A&M Dell Inspiron 9300, this is the C:\Rotomotion31 directory for software version 3.1). Using the command prompt in Windows in the Rotomotion software directory type:

```
Ground –s 10.255.0.5
```

The –s qualifier defines the ip address of the remote host with which you wish to connect. In the case of the Texas A&M autonomous helicopter system the ip address of the AFCS is defined as 10.255.0.5. The ground executable already has the remote port assigned as 2002 as a default, which is the assigned port on the AFCS, so no other qualifier needs to be added to the ground command to connect with the AFCS. The command line to call this function in Linux is very similar:

```
./ground –s 10.255.0.5
```

The Rotomotion executables on the Linux partition of the Inspiron 9300 are located in the Rotomotion/R3/linux-i386 folder. If one wishes to create a log file of the data collected by the AFCS during a flight, the command

```
./run-ground am001 –s 10.255.0.5
```

may be used, where am001 can be any file name you desire and will be the name of the log file created in the current directory. Another useful command is the query-msgs

command. This executable will display on the screen the different packets of information that are being received from the AFCS. The call line for this command in Windows is:

```
query-msgs –s 10.255.0.5
```

Again the –s qualifier is used to define the remote host ip address and again the same call may be used in Linux by adding a "./" in front of the command line in a manner similar to the ground station executable call. This command may also be used to read the messages being sent by the Wildfire microcontroller by typing

```
query-msgs –s 10.255.0.74
```

where 10.255.0.74 is the IP address of the microcontroller. At this time, the obstacle avoidance program on the microcontroller only stores one external client ip address at a time for sending duplicate controller messages. This means that it will only send command packets to the ip address that most recently inquired for messages from it in addition to the packets that it sends to the AFCS. This can easily be modified in the code to accept multiple remote inquiries later on if so desired by repeating the code used to remember and send to the one ip address. The query-state executable may also be used in a similar fashion to that of the `query-msgs` command with the –s 10.255.0.5 qualifier. The `query-state` executable only shows state information whereas the `query-msgs` will show all types of data packets. The `flyto` executable may also be called from these Rotomotion executables. A sample command of this is

```
flyto –s 10.255.0.5 -- 10 0 -5
```

where again the –s qualifier is used to define the remote host ip address. Other qualifiers may be used before the -- in the function call to define variables such as which coordinate system the call pertains to (absolute or relative, ned, xyz, or latitude-longitude). These definitions can be found by typing `flyto` –h at the command prompt for the help

information. The 10 0 -5 command in the default `flyto` executable (ie with no other qualifiers) will have the helicopter fly to a location 10 meters north, 0 meters east, and 5 meters up from the ground in the absolute ned coordinate system. The altitude is defined for the AFCS with a sign convention such that negative number for the down or z coordinate represents a position up or away from the ground. To exit any of the executables created by Rotomotion, press ctrl-c on the keyboard.

A demonstration of the interface and computer model of the helicopter flight has been provided by Rotomotion in the form of a simulation program. This program will simulate the AFCS by sending out data packets in a similar fashion and creating responses to commands with movements similar to actual helicopter movements based on mathematical models. This has proven to be a great proving ground for the obstacle avoidance software that has saved the helicopter from many crashes by testing programming logic in a harmless simulator environment. In order to use the simulator, several executables must be called in a particular order:

```
heli-sim
hover
heli-3d
ground
```

These four executables do not need qualifiers if they are all called on the same machine because they all default to the same ip address. If the commands do not need a qualifier then you can actually run them by finding the directory containing them using windows explorer and double clicking on each of the executables in order instead of calling them at the command prompt. You can run all of these executables on the same machine if it has a reasonably high amount of memory and processing power, otherwise you may want

to run the `heli-sim` and `hover` commands on one machine and the `heli-3d` and ground commands on the other machine. In this case the commands on the second computer would be

```
heli-3d -s 10.255.0.76
ground -s 10.255.0.76
```

where 10.255.0.76 is the static ip address assigned to the Ethernet port for the Inspiron 9300 when it is in Windows. The ip address for the Inspiron 9300 when it is in Linux is 10.255.0.3. This would mean that the heli-sim and hover commands are run on the Inspiron 9300 and the other commands are run on the other computer. The `heli-3d` program does not have to be run to utilize the simulation. It simply provides a 3-dimensional animation of the virtual helicopter as it is flying [4].

Wildfire Software

The program on the Wildfire microcontroller runs automatically once power is applied to it. If one wishes to stop this autorun execution, you must short pins 1 and 3 together on header CN8 (see Figure 11), then the microcontroller will default to the main

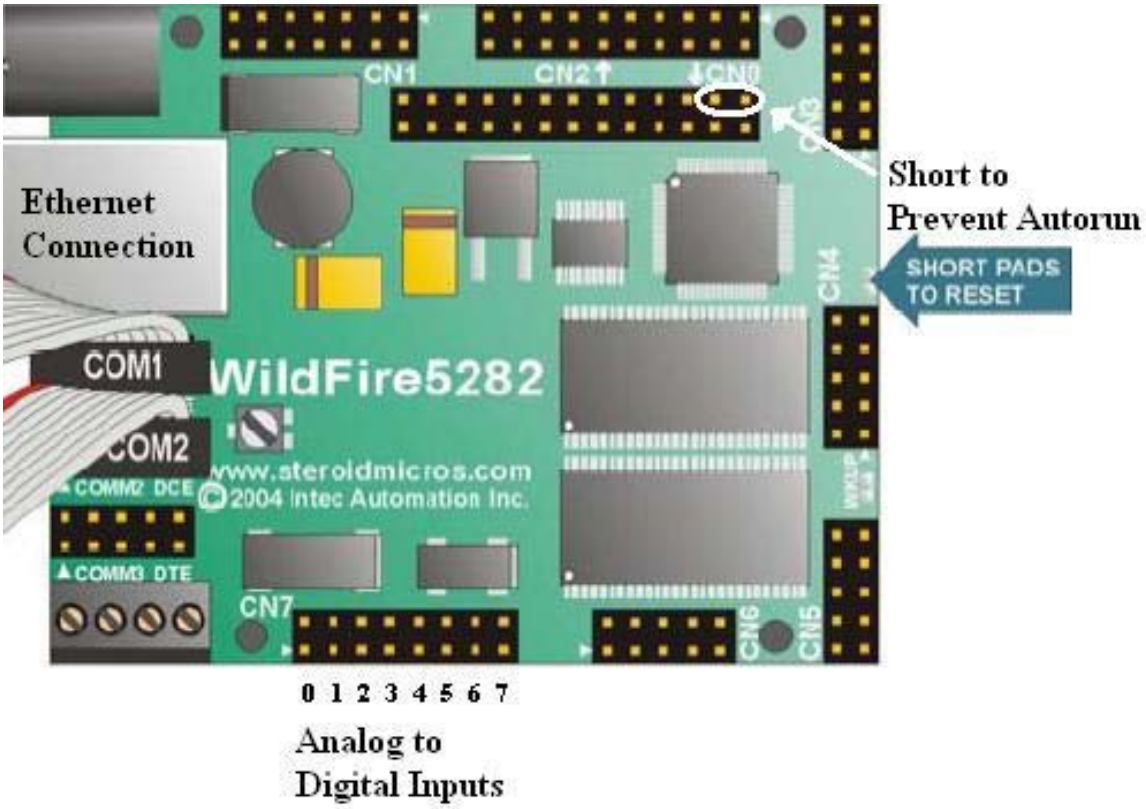

Figure 11: Wildfire Pinouts [8]

command prompt. Currently there are two main versions of the obstacle avoidance software: simulation and obstacle avoidance. As implied by the names, the simulation program contains code to simulate an obstacle and the sensor readings while the obstacle avoidance program contains code to read the sensor readings from the sensors. In case any modifications need to be made to this program I will go through the entire process of

compiling and loading the software onto the board. At the current time, the version of SBCTools provided by Intec Automation hangs up while trying to load software onto the board so this must be done manually. To do this, you must first save the file that you desire to compile and then build it using SBCTools. Once this is done you use the command prompt in Windows to go to the directory containing the desired project (for most computers with the SBCTools software this should be the C:\SBCTools\eclipse\workspace directory). In the workspace directory open the project folder you wish to load to the microcontroller and then open the Release folder within that folder. Once in this folder at the command prompt type

```
m68k-elf-objcopy -output-target srec test1.elf test1.s19
```

where test1 is the name of the compiled file that you wish to load onto the microcontroller. This command line converts the file to the s19 format, which is the desired format for the microcontroller. Once this is done, open hyper terminal in Windows and connect to the com port associated with the serial port that the Wildfire programming cable is connected to. Make sure the communication properties are set for 115200 baud, 8 data bits, 0 parity, and 1 stop bit. Once you are connected via hyper terminal to the powered on microcontroller, the dBUG> prompt will appear. This is the default command prompt of the Wildfire. Type

```
dl
```

and press enter. The controller will then ask you to start sending the file. To do this, click on the Transfer drop down menu at the top of the hyper terminal window and click on Send Text File. In the window that pops up change the Files of type: drop down menu from Text files (*.TXT) to All files (*.*). Next browse to the directory containing

the .s19 file you just created and click on that file and click Open. The program will then proceed to download the file to the microcontroller. Once the file transfer is done, type

```
go 0x10020000
```

at the dBUG> prompt to execute the file. To stop the execution if it is on a continuous loop either pull out the power cord and plug it back in or short the two pads located between headers CN3 and CN4 with a pen tip or piece of wire to externally reset the controller. This entire process should be able to be done by hitting two buttons in the SBCTools Eclipse front end once Intec Automation sends us a copy that will work with our computers. If you wish to load a program into the external memory so that it will permanently be saved onto the Wildfire type

```
dfl xf
```

instead of dl when you are at the command prompt to download the file to the external flash (xf). If an SD card is purchased for extra program storage, it can be inserted on the back side of the controller and can be saved to by typing

```
dfl sd
```

To execute programs from the external flash or sd card type

```
gfl xf
```

or

```
gfl sd
```

The autorun feature of the board will first look to the SD card for a program to run and then look to the external flash. To disable the autorun feature from a software standpoint type the lines

```
status
set autorun = no
```

This set command can also be used to turn off the watchdog timer and set the ip address of the Wildfire. More information on how to program the Wildfire microcontroller can be found at Intec Automation's website (www.steroidmicros.com) in their User and Programmer Manuals [9].

Obstacle Avoidance Program

UDP Communication

One of the main reasons the Wildfire microcontroller was chosen is because it had a built in Ethernet connection on the board and preprogrammed application program interfaces (API's) that dealt with UDP communication, which is the protocol selected by Rotomotion for their data communication. The source code for the API's that facilitate the use of Ethernet communication stems from the OpenTCP project that can be found at www.opentcp.org. As stated on the OpenTCP website, "OpenTCP is a project that brings a world-class, highly reliable and portable TCP/IP stack to the world of 8/16 bit microcontrollers." This is a free, open source C language based code that can be incorporated into microcontrollers and compiled on regular computers by including the header and library files in a C program preprocessor directive. The commands available for use with UDP communication include tcpip_init, tcp_tick, make_ip, IP<X>, RECEIVE_NETWORK_BUF(), RECEIVE_NETWORK_B(), udp_getsocket, the UDP Listener Function, udp_releasesocket, udp_open, udp_close, and udp_send.

Tcp_init is used in the code to initialize the TCP/IP subsystem. Once this is called, then udp_getsocket must be called to open a UDP socket for communication. When the socket is opened udp_open is called to bind that socket to a port and open it for communication. In the case of the obstacle avoidance software, the socket is bound to

port 2002, which is the same port the AFCS communication is bound to on its board so that the number is easy to remember. The function make_ip can then be used to convert the four separate numbers involved in an ip address into a single unsigned long integer so that it may be easily passed through various functions. Once the socket is open and bound to a port that has also been opened, then the device is ready for communication. To send something out from the microcontroller it must first be placed in the communication buffer in its entirety and then sent out as one entity. To accomplish this, a variable, generally called buf in the example code, is initialized as equal to the variables OTCP_TXBUF + UDP_APP_OFFSET. These two variables encapsulate the header necessary at the beginning of every UDP packet. After the header is placed at the beginning of the buffer the desired data may then be entered into it. This is done one byte at a time by stepping the pointer in the buffer up one and assigning that location in the buffer a byte value and then stepping the pointer again for the next byte. Once all of the desired bytes for the current UDP packet are entered into the buffer, then the udp_send command is called to send the packet in the buffer to a desired remote host ip address and port number. Once the information is sent, the tcp_tick function must be called to process TCP/IP packets. If incoming packets are found during a tcp_tick call that are of the UDP type, then the UDP Listener function is called within the tcp_tick function. This function may be defined by the programmer to perform whatever action they wish to the incoming UDP packet. To read the information from an incoming UDP packet, the RECEIVE_NETWORK_BUF function can be called to read all the bytes at once into a variable along with the total length of the packet in bytes into another variable. The RECEIVE_NETWORK_B function may also be used to read the incoming buffer

one byte at a time. The UDP Listener function also stores in variables the ip address and port from which the data was sent. The IP<X> function can be used to retrieve one of the four numbers in the ip address from the unsigned long integer that all four are stored in, where X is the position of that number in the address ranging from 1 to 4. Once all desired UDP communication is complete, the udp_close and udp_releasesocket functions can be called to close the socket for communication and release it back into the pool for reuse. All of these functions and more are described in detail with examples in the Intec Automation Wildfire Programmer Reference Manual that can be found on their website.

The format of the UDP packets that the Rotomotion system uses for communication is defined in the Rotomotion API User's Guide. The types of packets used for communication by the obstacle avoidance program include the open, ack, state, autopilot, flyto, and platform commands. All of the UDP packets for every type of Rotomotion API command start with a 12 byte header. This header is composed of an unsigned long integer that represents the time is seconds that the packet was generated by the AFCS computer, the following unsigned long integer represents the fraction of second in which the packet was generated. The third and final unsigned long integer in this heading represents the message id which corresponds to the type of message that is contained in the rest of the packet. All of the packet types are predefined so that once the message type is known the information in the rest of the packet can be interpreted in a consistent manner.

The first message type that must be used is the open message. This message is defined by the id being equal to the number one. This message must be sent to the AFCS by any device wishing to receive data from it. When the AFCS receives this packet it

records the port and ip address that it came from and adds those variables to its client list of addresses to send information to. If the AFCS does not receive an open packet from an address then no information will be sent to that address. Since the UDP communication protocol is connectionless (unlike the TCP protocol), there is no way of knowing if a packet that has been sent has actually been received by the desired host. In order to compensate for this issue, I created a conditional loop that continues sending an open packet to the AFCS until it receives an ack (or acknowledge) packet from the AFCS. The ack packet is defined by the id number being equal to the number two. An ack packet is automatically sent by the AFCS once it receives an open packet. In order to read and process packets that are received by the Wildfire microcontroller, I use the RECEIVE_NETWORK_BUF command to load the entire incoming UDP packet into the buffer. I then have the microcontroller identify and save the 12^{th} byte into a variable called id. The number stored in this variable corresponds to the message type as predefined by Rotomotion. The 11 bytes preceding this byte are unimportant time stamps for the first eight bytes and then zeros for the next three bytes as the message id numbers are all less than 256 and can therefore be defined by one byte even though four bytes are allocated for it. The microcontroller then compares this id to types that it is looking for using conditional statements and processes the packet accordingly based on its type. In the case of an ack packet from the AFCS, the UDP Listener function changes the variable connected that is causing the conditional loop sending the open packet to iterate so that that loop will end and the program will proceed. Since the ack packet is only meant to acknowledge an open command, there is no other information contained in the ack packet, and therefore no other processing is required [7].

One problem that I found in setting up this first form of UDP communication between the microcontroller and the AFCS was that even though the open packet I was sending was similar to the packets being sent by, for example, the query-msgs command from the laptop to the AFCS and followed the Rotomotion API definition, the AFCS was not recognizing the command. The AFCS did however recognize commands like flyto that contained more data. Upon discussing this problem with Mike Lavender, the software engineer at Intec Automation, he suggested that I use an Ethernet sniffer called Ethereal (www.ethereal.com) to analyze the UDP packets and compare them. Ethereal detects all packet types being sent to any port on the computer that it is running on and records the information in them as hexadecimal code. Upon analyzing the hexadecimal patterns in the UDP packets I was sending from the Wildfire and comparing them to the packets being sent by the `query-msgs` command, I found that the Wildfire was sending an extra junk byte at the end of my open packets. I found that if I added an extra zero to my open packet so that that junk byte would be zero instead of a random number then the open command would work. Discussing this matter with Mike at Intec, we decided that it must be a bug in the microcontroller programming that makes it have to always send 60 bytes in a UDP packet. The Rotomotion message types that did not have any data contained in them (ie open and ack) only contained 59 bytes (the 12 data bytes plus 47 bytes at the beginning of the packet containing the header that defined it as a UDP packet). With this issue resolved, I continued developing the code to interpret the other message types.

One analysis tool that I developed for the microcontroller software was to record the ip address of any computer that sent an open packet to the Wildfire ip address and

send duplicate copies of the commands sent to the AFCS to that address as well. With this feature I could type

```
query-msgs -s 10.255.0.74
```

on a laptop that was connected to the microcontroller and see the message types and contents of the messages as they would be interpreted by the AFCS. This helped to ensure that the numbers were being encoded right and being outputted in the right order. It also allows the user to see exactly what course of action the obstacle avoidance program on the microcontroller is prescribing to deal with detected obstacles and changes in the velocity direction. On a single ground station laptop, the ground program can be run to connect with the AFCS while the `query-msgs` program can be run simultaneously to connect with the microcontroller and both data streams can be fed through the wireless data link from the helicopter to the ground.

Floating Point Mathematics

The next major obstacle in parsing and interpreting message packets arose when trying to send or receive any packets containing a variable defined as a double precision floating point variable (such as those contained in the state, autopilot, flyto, and platform commands). The problem arose when I tried to input or read a double from a packet. The microcontroller did not recognize those numbers. Upon discussing this matter with Mike at Intec, he revealed to me that the Freescale Coldfire MCF5282 microcontroller that powers the Wildfire did not have a floating point unit and could therefore not handle floating point variables. This being the case I had to develop a new number system that could keep track of floating point numbers using only integers without losing the precision of the numbers that were located after the decimal point. Mike had suggested

using a long integer (utilizing 32 bits so that it could range from -2147483648 to 2147483674) accompanied by character (or 8 bit integer) that would specify the decimal place location within the number. Considering this I decided that this could method could easily diminish number precision as the digits preceding the decimal began to grow and could easily cause overflow errors as these 10 digit numbers were added or multiplied. I instead decided to create a number system defined by an unsigned long integer to represent the digits preceding the decimal, another unsigned long integer to represent the digits following the decimal, and an unsigned 8 bit integer to represent the sign of the number. I had originally hoped to contain the sign in the variable representing the numbers preceding the decimal place, but ran into loss of sign difficulties when that number became zero while the variable for the digits located after the decimal remained a nonzero number. This three variable combination only uses 8 more bits of memory for storage (long integer (32 bits) + long integer (32 bits) + 8 bit integer compared to 64 bits for a double precision floating point variable).

In order to convert to and from the double format being used by the Rotomotion controller, I had to ask Dennis D'Annunzio what convention they were using. He pointed me to the website, http://www.psc.edu/general/software/packages/ieee/ieee.html, which has a definition for the IEEE Standard 754-1985 for binary floating point arithmetic that they used for their doubles. Based on this standard, the double precision floating point variable is equal to

(-1)^S*2^(E-1023)*(1.F)

where S is the first bit , the next 11 bits represent the binary form of E, and F is the remaining bits of the 64 bit variable turned into a binary fraction where the first 1

represents 1/(2^1) and the second 1 represents 1/(2^2) and so on to where the last one represents 1/(2^52). To facilitate the conversion from this format to the all integer convention I created, I made a function called buftodec which receives an array of eight 8-bit numbers representing the 64 bits of one double variable. A peculiar thing about these eight 8-bit numbers that are sent by the AFCS is that they are sent in reverse order to where the first 8-bit number represents the last 8-bits of the 64-bit binary representation of the double and so on. The order of the bits within each byte is not in reverse order, only the bytes themselves within the 8-byte double. This anomaly was found again by using Ethereal and analyzing the actual bytes sent by the AFCS and comparing them to the numbers that they were supposed to represent. It was also found that the controller only does this for doubles. Integer variables greater than 255 are sent in normal binary order in which the last byte of a 4-byte long integer represents numbers from zero up to 255 and the next byte to the left provides the next eight higher binary decimal places to facilitate the creation of numbers from 256 though 65536 if you apply an 8-bit binary shift to the right to that byte. This relation is also true for the other two remaining bytes. The reversal of the order of the bytes for the buftodec function is actually done to the numbers as they are saved into the array of chars before that array is passed to the function. The function then applies the IEEE standard for doubles in a bitwise manner to the 64-bits of data and outputs that number as the global variables genericint, genericdec, and genericsign, where genericint represents the digits before the decimal place, genericdec represents the digits after the decimal place, and genericsign represents the sign where a value of 1 represents a negative number and 0 represents a positive number. This convention using int, dec, and sign is followed throughout the

program with different variable names being used in the place of generic. Due to this issue with doubles, I also created another function called dectobuf that would take the int, dec, and sign variables passed to it and output an array of eight 8-bit numbers in the proper order so that it may be written into the buffer and sent via the udp_send command and interpreted by the AFCS or ground station as the proper corresponding double precision floating point variable.

The problem that arises in creating a new number system is that the simple arithmetic functions must then be redefined as well to accommodate this new convention. The simple acts of addition and multiplication cannot be used because an entire number is actually defined by three different variables. In order to accomplish this, I also wrote functions called decintmult and decintadd. The decinmult function actually breaks the decimal part of the each variable into two halves so that the entire decimal portion of the number can be multiplied together without creating an overflow error and without losing any decimal places in precision. Each half represents four decimal places of the eight digits that come after the decimal point so that, when multiplied together, their product will be no greater than 99800001 (9999 x 9999), which is well below the maximum value of an unsigned long integer, 4294967295, preventing an overflow error that would crash the program or lose data. This brings the overall total up to three different actual numbers per double that have to be multiplied together, adjusted to their proper decimal place, and recombined to form a new number. Any portion of the decimal portion of the number that is above 99999999 must also be added to the integer portion and then removed from the decimal portion to adjust for the transition of digits from the right of the decimal point to digits to the left of the decimal point. The sign of the product is

determined by conditional statements that apply the rule that a positive number times a positive number equals a positive number, a negative number times a negative number equals a positive number, and a negative number times a positive number equals a negative number. The resulting number is then returned to the program in a similar fashion to the buftodec function as the global variables genericint, genericdec, and genericsign.

The decintadd function has to overcome similar difficulties inherent with the new number system, and, while the decimal portion of the number does not need to be broken up as in the case of multiplication, there are many more special cases to consider in addition. The special cases generally have to deal with the sign of both of the numbers and whether one is greater than the other. If the signs of the numbers are the same, then the addition is pretty straight forward. The only extra task that must be carried out is adding one to the integer portion of the number and subtracting 100000000 from the decimal portion of the number if the decimal portion of the number is greater than 99999999. The resulting sign is the same as the sign of the original two numbers. If the sign of the number to be added are opposite, then many special cases must be considered based on which integer portion of the number is greater and that will dictate actions determined by which of the decimal portions is greater. The decintadd function may also be used for subtraction by reversing the sign of the number that is to be subtracted before entering it into the function call.

Unfortunately, unlike addition and multiplication, division is not commutative, and therefore cannot be broken up into parts like the addition and multiplication operations were in the decintadd and decintmult functions to facilitate use of the new

number system while maintaining accuracy. Therefore, in order to perform division, the numbers must be turned into two single unsigned long integers. This can be done by multiplying the integer portion of the number that would be the numerator by the highest power of ten that would keep it below the maximum value for an unsigned long integer and dividing the decimal portion of the number that would be the numerator by ten raised to a power equal to eight minus the power of ten by which the integer portion of the number was multiplied. The powers of ten for multiplying and dividing the integer and decimal portions of the denominator, respectively, would then be determined by a compromise between how many decimal places the user was interested in maintaining in the denominator number and how many they wanted in the result. For example, if the numerator integer portion were multiplied by 100000000 and the denominator integer portion were multiplied by 10000, then the result would have four decimal places of accuracy. If the denominator integer portion were multiplied by 1000, then the result would have five decimal places, and so on.

Still another byproduct of not being able to use floating point variables lies in the inability to use the sqrt and trigonometric functions in C, which both rely on that data type. To bypass the sqrt function I created a function called squareroot that finds the square root of the number passed to it using an exhaustive iteration process where it starts with the thousands place and increments that decimal place by one, multiplies the incremented number by itself and compares that to the number whose square root it is trying to find. Once the number squared is greater than the given number, then the decimal place that was last incremented is decremented one and the process repeats for each decimal place down to .00000001. Once the iterations are completed, the square

root is returned as the generic variables. Since trigonometric functions cannot be used, the program determines angles of vectors by utilizing the x and y (or north and east) component lengths of the vectors to find the proportion between their lengths. This is done by dividing the length of the y (or east) component by the sum of the lengths of the x and y (or north and east) components. This proportion, when multiplied by Pi/2 (or 90º), will give the angle of the vector from the x (or north) axis. If the x (or north) component of the vector is negative, then the calculated angle must be subtracted from Pi to give the actual angle deviation from the positive x (or north) axis. The sign of the y (or east) component must then be used to determine the sign of the angle. If it is positive, then the angle is positive (according to the sign convention set forth by Rotomotion), and if the sign of the y (or east) component is negative, then the angle sign must also be negative. To create proportional vectors (for instance if you desire to fly to a point in the same direction that you are currently flying, but with a shorter magnitude), the program can determine the proportion of the desired distance to the previously desired distance (where both distances would be calculated using the Pythagorean Theorem) and then scale down the north and east components of the previously desired distance using that proportion and send it using the absolute flyto command. This was done as a coordinate transformation method when a glitch in the Rotomotion simulation software was not allowing for proper response to relative flyto commands based on the xyz helicopter body frame coordinate system when the helicopter was supposed to stop after detecting an obstacle.

I/O Sub Functions

Once the subtle nuances were ironed out in dealing with the lack of a floating point unit on the microcontroller, I then proceeded to write two more functions to deliver the flyto and lookat UDP messages to the AFCS. The flyto function is passed the variables to determine if it is actually a flyto command or just a failsafe, the position mode, the heading mode, the desired position in ned (or xyz coordinated depending on which position mode is selected), and the desired heading in radians. The flyto and failsafe UDP messages are the same in format and length and only differ by their id number. The purpose of the failsafe is that if a failsafe time is set and the controller does not receive another flyto command or failsafe packet within that failsafe time, then the controller will stop pursuing its current goal and fly to its failsafe position instead (ie a stable hover in place). This is a safety feature that helps to keep the helicopter from flying to unwanted positions if communication is lost. The lookat function is passed the desired pitch and yaw angles of the camera platform, which it then sends via a platform command to the AFCS. The purpose of both of these functions is to facilitate the sending of desired variables via UDP packets. Since the flyto and platform commands are used repeatedly in the body of the obstacle avoidance program, I thought it best to consolidate the process of converting the numbers to the proper format and placing them in the proper order with proper the padding to functions outside the main loop. These functions also provide the redundancy necessary when using connectionless UDP packets to keep sending the current command until it is recognized by the AFCS.

In addition to the UDP communication APIs, a few others are necessary to exploit some of the other tools available on the Wildfire microcontroller. The other main

important feature that made the Wildfire a viable option for this experiment is its analog to digital (A/D) inputs. These inputs are necessary to read the analog voltage that is output by the Senscomp sensor in proportion to the distance that it is measuring. Intec facilitates the exploitation of these A/D inputs by providing two APIs to initialize and read the values from these inputs. First, to initialize the A/D inputs, a call must be made using the ad_init function. This function initializes how the A/D inputs are going to work based on sampling frequency and sampling mode. For their use in this application, I set the sampling time to the longest available, which is once every 8 μs, and I set the mode to average the past eight measurements together to determine the reading. I chose these parameters based on the fact that the sensors can sample at a maximum rate of 10 Hz, which is relatively a much slower sampling rate, therefore only the slowest rate available from the A/D converter is needed. Once the inputs are initialized, then the value of each individual port is read by the ad_get function, which is passed a single integer parameter which specifies which channel is to be read. Another feature governed by APIs available on this microcontroller is the watchdog timer. This is a safety parameter you can set in the board so that if the program freezes up, the board will automatically reset itself, which will restart the program if the board is set to autorun mode. The watchdog timer is set by calling the wdt_timeout_set function and passing it a variable based on how long the timer should wait between “feedings” before it resets the board. In the case of the obstacle avoidance software, the loops run fairly fast, so I set the timeout value to approximately half a second. The watchdog timer is “fed” by calling the function wdt_feed. These “feedings” reset the timer each time they are called, preventing it from

reaching the preset timeout and resetting the board. Descriptions and code samples of these APIs are also available in the Intec Automation Programmer Reference Manual.

Sensor Simulation Sub Function

In order to test the logic, interface, and reaction of the obstacle avoidance software without actually flying the helicopter, which would risk damaging the helicopter and harming onlookers, I needed to create code in the obstacle avoidance software to simulate the distance measured by the sensors based on a simulated obstacle. To accomplish this I created another function called sensorinput in the program in which a simulated obstacle could be defined and which could be called with syntax similar to that used to call the functions to read the actual sensors. In this function, the obstacle can be a square (representative of a building) of any size in any location and is defined in the first few lines of the function as xfrontobs, xbackobs, yleftobs, and yrightobs. The location and size of the object can be changed by altering the values of these variables. The only variable passed to sensorinput is the location number corresponding to the sensor whose value the main program is asking for, which is the same number convention the ad_get function uses to read the real sensors. The function then determines the direction in which the desired sensor is pointing based on the current yaw of the helicopter, the current yaw of the camera platform, and the sensor angle relative to the center sensor (sensor number 2). If this angle direction is not between –Pi and Pi, then it is converted to the proper format. The function then determines if the helicopter is between or beyond the east and west planes or the north and south planes of the obstacle. If it is between or beyond the east and west planes, then the east-west distance is not calculated because based on the current velocity direction of the helicopter it will not encounter either of

those sides. Similarly if the helicopter is between or beyond the north and south planes, the function will not calculate the north-south distance from the helicopter to the obstacle. This qualifying process eliminates unnecessary calculations and increases performance of the program. If, for example, the north-south distance calculation does need to be performed to determine the distance from the south wall because the helicopter is south of the obstacle flying in a northerly direction, the function first determines the distance in the north-south direction between the helicopter and the south wall. The function then uses that distance with the given angle to determine the magnitude and direction of the east-west component of the distance. This is then added to the current east-west position of the helicopter. If the resulting sum is between the values of the east and west walls of the obstacle, then the distance magnitude is calculated by plugging the east-west and north-south components of the distance into the Pythagorean Theorem. If this distance is less than 1219, which is the maximum distance that the actual sensors can read in centimeters, then the calculated distance is returned by the function as the sensor distance reading, otherwise the distance reading is kept at 1219 as with the actual sensor to represent that no object is detected.

Obstacle Avoidance Algorithm

The main body of the program begins by initializing the UDP port for communication and sending open commands to the AFCS until an ack packet is received. Once this is done, then the A/D inputs are initialized and the main loop begins. The first part of the main loop is dedicated to pointing the sensors in the direction of the velocity of the helicopter so as to detect obstacles that could most imminently cause a collision. This is done by using the velocity components of the helicopter as measured by the

AFCS in the x and y direction defined by the body frame of the helicopter. These values are plugged into the proportional method for finding angles as described earlier in this paper to determine the resultant direction of the velocity. This resultant angle is then sent to the AFCS via the lookat function to point the camera platform in that direction. In addition to the x and y components of the velocity, which only dictate the yaw component of the platform angle, the pitch angle of the helicopter is measured and the negative of that angle is fed to the lookat function as the desired pitch angle of the platform to continually keep the platform level with the ground. This calculation to point the platform in the direction of the velocity is only made if the sum of the x and y components of the velocity is greater than one. This is done to prevent sporadic movements of the platform based on small fluctuations in the velocity when the helicopter is in a stable hover. If the sum of the velocity components is less than one, then the platform is simply pointed straight ahead, although the pitch is still continually adjusted based on the pitch of the helicopter.

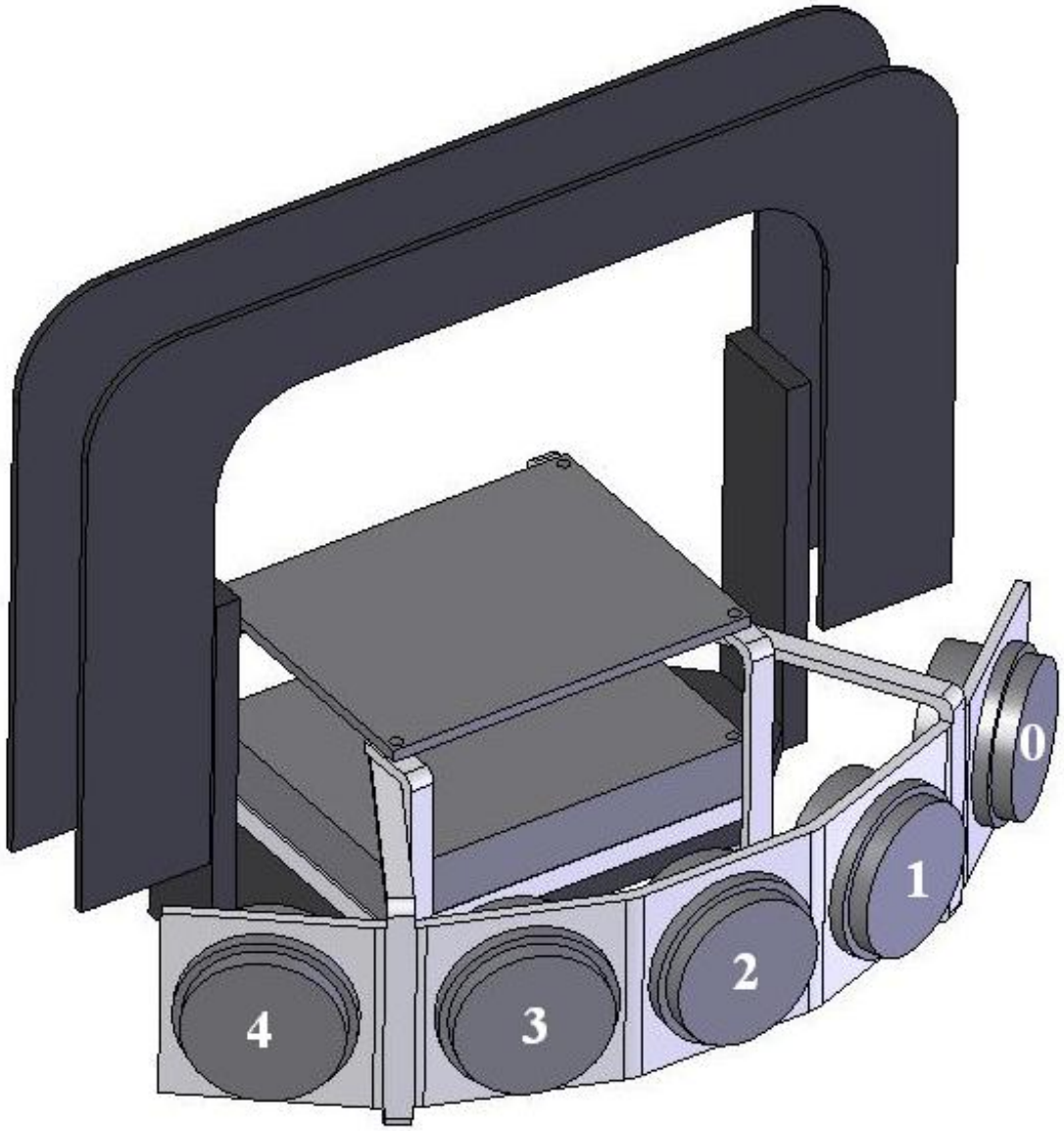

Figure 12: Sensor Reference Numbers

With the sensors facing in the direction of the velocity, the value of the sensors must now be read to check for any obstacles. This initial reading is then filtered through various layers of reactionary scenarios based on which sensor sees an obstacle and the distance at which that obstacle is detected. The first and most important comparison is based on the middle sensor (sensor 2). See Figure 12 for the sensor numbering convention. This sensor is pointed directly in the path of the velocity of the helicopter

and will therefore sense the obstacles that present the most prevalent danger to the craft. If this sensor detects an object less than 11 meters away (a distance slightly below the maximum value of the sensors to ensure proper detection of the object) then the program checks to see if the sensors to the immediate left and right of the center sensor see an object. If both of those sensors see an object, then that means that there is an obstacle in a plain fairly perpendicular to the direction of travel of the helicopter that is directly in front of the helicopter. If this is the case, then the program first records the presently desired position and heading and then takes another reading from the center sensor and uses that value to generate a flyto command that will stop the helicopter four meters from the obstacle. Once the helicopter has stopped, the algorithm checks the values of sensors 1 and 3 (the ones located to the left and right of the center sensor, respectively) to see which distance is greater. If the values are the same, then the platform is already facing in a direction perpendicular to the plane of the obstacle. If one value is less than the other, then the program sends a command to turn the platform in the direction of the sensor indicating the shorter distance until the distances are equal or the distance that was originally shorter becomes the larger of the two. The angle of the platform at which the two sensors equalize is then determined to be the direction perpendicular to the plane of the obstacle. Utilizing this angle of the platform in conjunction with the yaw angle of the helicopter, the program turns the helicopter to a direction parallel to the plane of the obstacle either to the right or left based on which direction provides the velocity vector most closely related to the initially desired velocity direction. Once the helicopter has turned so that it is parallel to the plane of the obstacle, the sensor platform is then turned in the opposite relative direction at an angle of 50^{o} relative to the heading of the

helicopter so that one sensor is still facing in the same direction as the helicopter, which will be the direction of the velocity as it is following the wall, and the two sensors on the opposite side of the array will by facing the obstacle to continually monitor the distance between the helicopter and the obstacle. The program then commands the helicopter to fly forward five meters along the wall. When the helicopter is within a couple meters from its new desired location, it checks the distance between itself and the obstacle again and modifies its path to a direction further away from the wall if the wall is less than six meters away, or closer to the wall if the wall is found to be more than six meters away. This adaptation allows for small discrepancies in the indicated plane of the wall as well as changes in the plane of the wall. The program iteratively sends commands to proceed along the wall in five meter segments, adapting its path according to the wall distance as it goes, until the wall is no longer detected. Once the wall is no longer detected, the algorithm sends the helicopter an additional five meters in the direction in which it was traveling to ensure clearance from the obstacle. The helicopter is then turned back to the direction in which it was initially traveling and is resent the commands that it had received before it encountered the obstacle to continue on the originally desired path. This process can continually iterate if another obstacle is detected, including another wall of the same obstacle.

If an obstacle is detected by the center sensor and then only by one of the two sensors immediately to either side of it, then a different course of action will be taken. This scenario indicates that either the helicopter is flying toward the corner of an object such that the path is clear to one side, but not to the other, or that the helicopter is flying toward an object in a direction that is not perpendicular to the plane of that object. In

either case, the proper course of action is to modify the flight path toward the direction where the obstacle was not detected and then take another sensor reading to reevaluate the scenario and react again accordingly. To accomplish this, the current desired position and heading are recorded and a flyto command is sent to the AFCS modifying its path to a distance two meters in the direction it was initially heading and five meters in the direction opposite to side in which the obstacle was detected. Once this maneuver is complete, the program resends the initial desired position and heading and reenters the main loop to check the sensors again a reevaluate its situation.

If the middle sensor continues to detect an obstacle and the sensors to either side of it still have not detected anything by the time the middle sensor is within eight meters of the obstacle, then it is determined that the obstacle is narrow (less than 2.8 meters wide based on trigonometric calculations). In this case, the program sends a flyto command toward the object at the distance detected and five meters to the side of that object. Again, once that destination is reached, the initial desired position is then reacquired and the loop continues.

The final scenario is based on an obstacle being detected by either of the outermost sensors while nothing is detected by the center sensor. This situation dictates that an obstacle has been located to the side of the helicopter. If either of the outermost sensors detects that the obstacle is less than eight meters away, then the program begins to calculate the difference between the magnitudes of the y components of the distance measurements between the outermost sensor and the sensor next to it. The program then uses these distances to determine the plane of the obstacle detected. Once that plane is the determined, the program sends a flyto command based on a one meter forward

displacement and a sideways displacement away from the object based on the proper proportion that would give the helicopter the proper direction of motion so that it would be moving in a plane parallel to the plane of the obstacle. This process once again continues iteratively until the obstacle is no longer detected and then the initial desired position and heading is reacquired and the program is released back to the main loop.

TEST RESULTS

A few initial flights were required in full manual mode without the AFCS powered or connected to the servos to check the set up of the helicopter and ensure that it was flight capable without introducing the highly uncertain initial variable of an untuned autonomous flight control system. The initial flights were piloted by Anthony Jager who also provided his flight mechanic expertise to adjust the helicopter so optimize its flying abilities. The main problem we encountered on the first flight attempt was getting the engine started for the first time. After many attempts at a pull start with the choke in the on and off position and adjusting the mixture screws the engine would not start. Discussing the problem with Chris Bergen from Bergen RC helicopters we found that the mixture screws should be no more than about 2 ½ turns out. Comparing this to the initial setting of the carburetor we found that we were using too rich of a mixture and it was flooding the engine. Making this adjustment on our next flight attempt we were able to start the engine and perform a successful flight. Anthony flew the helicopter only a few feet of the ground during initial tests to minimize the potential damage to the helicopter if something went wrong and it crashed. Once the helicopter flew successfully we added the additional landing skid containing the AFCS and D-Link wireless router to the bottom of the craft. Anthony stated that the helicopter flew sluggishly mainly due to the extra payload weight from the AFCS and other electronic devices on the helicopter. We did experience loss of control in flight where Dave Lund, acting as the safety pilot, did have to put the helicopter down as Anthony had lost control through the test pilot receiver.

With a successful test flight on record for this helicopter, we scheduled a meeting with the people at Rotomotion at their office is Charleston, South Carolina to adjust the

control loop gains in the AFCS. Josh Davis and I transported the helicopter from College Station to South Carolina where Dr. Reza Langari met us for the test flight. The test flight was performed at the hobby RC helicopter club's flying field in Charleston with Dennis D'Annunzio Jr. acting as the test pilot and John Dornish running the ground station. During the first flight we found that we again lost test pilot control during flight, this time with the helicopter at an altitude of about ten feet, and had to kill the engine using the safety pilot control. This action resulted in a fairly hard landing that bent the aluminum tubing on the landing skid. With this problem prevalent and unpredictable, Dennis and John decided we should run a strap down test and try to determine the cause of this loss of control. We strapped the helicopter to a long folding table and started the engine up. After running for a short period of time, control was again lost through the test pilot remote and the helicopter had to be killed using the safety pilot. Dennis and John postulated that the loss of control was caused by an electrical potential difference created by static electricity build up created by the nylon belt drive running through the aluminum tail boom as they had seen this problem before in their dealings with Bergen Observers. To alleviate this problem we checked our ground wire scheme that we had already put in place on the helicopter per recommendation from Rotomotion before we came to South Carolina. The ground wires that went from the triple bearing block to the aluminum landing skid and from the landing skid to the mounting plate for the AFCS were removed as Rotomotion stated that they found that that particular connection could cause more problems than it would alleviate. We then checked the electrical connection between the tail boom, triple bearing block and main rotor bearing block. Problems did arise from an excess of Loctite preventing electrical contact between the grounding wire

that the bearing blocks. This was alleviated by continually adjusting the screw until electrical contact was made. The multimeter should be able to detect electrical contact between the main rotor shaft, the drive shaft going through the triple bearing block and the portion of the tail boom where the annodization was grinded off to allow for electrical contact. Proper grounding cannot be determined from checking only the screws where the grounding wires were attached or even the bearings at the grounding locations.

Another measure we took was to move the receiver box from its location on the upper battery tray just below the tail boom in the rear of the helicopter to a location on top of the D-Link wireless router on the AFCS mounting plate in the aluminum landing skid towards the front of the vehicle. This would distance the test and safety pilot receivers as well as the safety relay board as far away as possible from any EMI that might be created by the tail boom. Once this modification had been completed, we again attempted a series of strap down tests to determine if the control loss problem had been eliminated. The tests showed that we were now able to maintain control of the craft via the test pilot remote.

The strap down tests also revealed a problem that we had not been able to observe in our previous flights. As the test went on, the RPMs of the motor began to increase without input from the test pilot remote to increase the throttle. After shutting down the engine, John Dornish found that the clutch on the helicopter was starting to get extremely hot, which indicates that it was most likely starting to slip. This slippage would explain the unexpected RPM increase as the load to the motor would decrease as slippage in the clutch worsened during the test. In an attempt to diagnose and rectify this problem we had to remove the clutch from the helicopter. This is accomplished by first dropping the

motor from the main body of the craft by removing the four screws and nyloc nuts that mount the motor tray to the body frame. With these fasteners removed, the engine can be lowered to sit in the aluminum landing skid. This skid does not need to be removed to access the clutch as enough room is provided by simply allowing the engine to rest on the top of the landing skid. To remove the clutch, the set screws on the starting cone must be loosened and the cone removed. After this is done, the clutch should be able to be removed from the clutch bell. Six of the eight screws mounting the triple bearing block to the frame must then be removed so that the triple bearing block can freely rotate enough to take the tension off of the nylon belt that goes to the tail rotor which is the last thing holding the clutch in place. After the tension to the belt is loosened, the clutch bell can then also be removed from the helicopter. The lining of the clutch bell and the surface of the clutch did show signs of slippage as the color had turned brown indicating burning due to the high heat created by friction from the slipping between the clutch and bell. In attempt to counteract this problem we cleaned the clutch and bell with alcohol to remove any dust that had been created by the slippage that might be facilitating more slippage and improve potential for better contact between the clutch and bell. Once we had cleaned the clutch and bell we had to put them back into the helicopter in a manner that is the reverse of the removal procedure. A major consideration when putting the triple bearing block back into position is to ensure the proper contact distance between the gear teeth connected to the clutch bell and those connected to the main rotor shaft. Dennis said the best way to accomplish this was to place a piece of paper between the gears while securing the triple bearing block into place. This will provide the proper gap distance between the gear teeth to prevent binding of the gears from too tight a

connection or breaking of gear teeth from too lose of a connection. Once the triple bearing block is secured back into place, the tension on the drive belt to the tail rotor should be checked to make sure that it will not slip either. When mounting the engine back into place one should also make sure that the opposing flat surfaces from drive portion from the engine and the clutch are as close to exactly parallel as possible to prevent binding and that the drive pins are pushed as far into the clutch as possible to maximize load dispersion through those pins to prevent wear and breakage.

Once the clutch was placed back in the helicopter, we performed another strap down test to see if the clutch was going to continue to heat up indicating that it was continuing to slip. The clutch did not heat up to the extent that it had previously so we decided to do another test flight. For this test flight we recalibrated the magnetometers and tried a flight using the gains that Rotomotion had from previous flight with their own Bergen Observer that they had done. Dennis decided to try to put the helicopter in full autonomous mode right away instead of slowly stepping up the levels of autonomy to determine if it would function properly. He said this would save time because we would know if the gains were good to use or not from the start. If the helicopter did not act properly when he hit the switch, then we would know that some tuning adjustments would have to be made to the gains that we had set in the AFCS. Again during this flight Dennis was the pilot and John manned the ground station. Dennis controlled the helicopter to take off and, once he had it in a stable hover, he flipped the switch to enable the AFCS to take control and perform an autonomous hover. The helicopter hovered autonomously for a few seconds before it started losing altitude. Dennis retook control of the craft once he felt that it was not holding the desired position well enough. He then

tried to increase the helicopter's altitude manually and found that it was having difficulty keeping in the air. He determined that the helicopter did not have enough power to lift itself off of the ground with the additional weight that was inherent in the landing skid, AFCS and wireless router. He gave us a Zenoah G26 engine to replace the Zenoah G23 engine that was on the helicopter in the hopes that that might help alleviate our problems with being underpowered.

Before the next test flight I swapped out the G23 engine for the G26 engine. I did this by removing the four screws and nyloc nuts that connect the engine tray to the helicopter frame. In order to remove the engine from the frame I had to also remove the four screws that attach the helicopter frame to its landing skids. This allowed for enough room for the engine to be pulled out from the bottom of the craft. The G26 engine has a slightly larger bore than the G23 but has the same form factor and size so that it can easily fit into the same space as the G23 and uses the same holes for mounting. I mounted the new G26 engine into the frame using the same screws and nyloc nuts from the G23 engine and then refastened the helicopter frame to its landing skids to complete in engine swap. Out next scheduled test flight with Anthony Jager got rained out. Since it seemed that it would be a little while before we got a chance to fly again I went ahead and modified the helicopter further by replacing the D-Link wireless access point with the much higher powered Renasis access point that we had purchased and integrated that access point with a wired router and the microcontroller that I had selected so that they would fit in one box. This integration involved removing the access point and wired router from their enclosures, soldering power leads to all three boards and modifying an enclosure so that all three boards would be mounted securely with access the Cat5E ports

available from the outside of the box. Each of these components was selected because, among other beneficial qualities, they all used 12V inputs. This would eliminate the need for the separate battery that was required to power the D-Link access point which, in conjunction with reducing enclosure weight by combining them into one enclosure, saves precious weight on the helicopter.

With successful test of the new electronic equipment and Anthony again having time to come out and fly we were ready for another test. When Anthony arrived, we started going through the preflight safety procedures. During the range check with the test pilot transmitter we found that we were getting a very poor quality of service and loss of control at an unacceptably short distance from the helicopter. The helicopter was also having difficulty acquiring satellites. We decided that it would not be safe to fly so we had to let Anthony leave, which was unfortunate as this was his last opportunity to fly for us because he was moving back to Dallas. We had hoped to get another pilot out while Anthony was there to observe Anthony flying and have the chance to ask him questions, but we were unable to get anyone out at that time. I went about troubleshooting the aircraft to solve the communication problems we were having and after extensive experimentation with EMI shielding positioning found that the wireless access point antenna's close proximity to the PCM output from the modified receiver was to blame for the communication drop outs we were experiencing with the test pilot transmitter. This problem was easily eliminated by rotating the antenna of the access point so that it pointed downward instead of upward creating a sufficient amount of distance between the antenna and the PCM cable while still maintaining the proper polarization direction of the antenna signal. I also discovered through experimentation and discussions with John and

Dennis at Rotomotion that GPS units tend to have difficulty acquiring if there is a large amount of RF noise in the GPS frequency range. I found that the wireless access point provided such a level of RF noise. To overcome this problem, it is necessary to start the AFCS and allow it to run for a few minutes to acquire satellites before the access point is turned on. Once the GPS has acquired the satellites, the RF noise does not affect the GPS performance or cause it to lose any of the satellites that it has found.

With the communication issues solved, we continued about the task of finding a new pilot. Roger Wagner, the owner of the American Aerospace hobby shop, suggested Pat Cole. I contacted Pat and arranged for him to come out for a flight. Pat ended up not showing up so I contacted him again and he explained his absence and we arranged for another test flight, which he again did not show up to. At that point I turned to Dave Lund to see if he would have any more success communicating with Pat and getting him to come out for a flight. Dave arranged another test flight with him and again he did not show up so we decided to pursue other options. We decided to use Curtis Youngblood, a pilot who has been repeatedly named the world champion r/c helicopter pilot and who by chance lives in College Station. We had hesitated to use him before because he had warned us that his time was very limited and he would only be available for a few flights. As weeks had gone by without flying the helicopter and all of our other local options did not work out we decided that we would do our best to prepare the helicopter for each flight and use our limited time with Curtis carefully.

Before out first flight with Curtis I decided to further test the obstacle avoidance software that I had been writing and that had successfully avoided obstacles when interacting with Rotomotion's simulation software by pushing the helicopter around on a

cart and reading its reactions to obstacles in its path. The first discrepancy I realized was that the ultrasonic sensors, while able to read walls at angles inside that office where I was testing them, was unable to detect obstacles that had a plane at any other orientation from the sensor face than very close to parallel. The only other major difference between running the obstacle avoidance algorithm in simulation and in real world situations was the delays required in the program while it waited for the sensor platform rotate over larger angles. After modifying the program to take into account the changes, both in the real world version of the software and in the code that simulates the sensors in when the algorithm is run in simulation, I once again pushed the helicopter around on a cart outside and it successfully detected and sent the proper commands to fly around a trailer that I had set out as an obstacle.

With this test successfully completed we had Curtis out for our first test flight with him. All the safety and communication checks were successful so we proceeded with trying to perform autonomous flight. Curtis initiated a take off and flew the helicopter around manually to get the feel of it. After he was comfortable with it, he set it up in a stable hover and flipped the switch to transfer control over to the AFCS. The helicopter again demonstrated flight characteristics similar to those we saw in our last flight with Dennis in South Carolina. The helicopter could not hold the altitude and there was an audible increase in RPMs from the engine as the helicopter lost altitude. These symptoms pointed to the probability that the clutch was again slipping inside the clutch bell. Curtis and I took the clutch out of the helicopter and saw more signs of charring indicating more slippage in the clutch. Curtis recommend scoring the contact surfaces of both the clutch and clutch bell to provide some level of friction as the charring on both

surfaces tends to decrease their contact friction creating more slippage. We scored the clutch by tapping it with a flat head screwdriver and scored the clutch bell lining by using a small razor blade. Once Curtis was satisfied level of scoring on both objects we reassembled the helicopter and attempted another flight. This flight turned out better with regard to power and ability to maintain altitude and actually allowed helicopter to perform an autonomous hover for more than a minute before it started losing altitude as the clutch started slipping again. Deciding that scoring the clutch and bell was nothing better than a temporary solution we sent both items back to Bergen for replacement and repair. Bergen was able to replace the clutch, but the clutch bell for that particular model of Observer was no longer in production so the had to reuse that part and just redo the lining.

When we received the clutch back from Bergen we called Curtis out for another flight test. The clutch and bell appeared to be in much better condition more apt to function properly so I reinstalled it before Curtis arrived for the flight. When we tried to start up the helicopter this time, the rotor head seemed to try to turn much harder than when we had previously tried to start the helicopter. This was an indication that perhaps the clutch was engaging too easily as opposed to not engaging fully as was the previous problem. Fortunately this issue does not pose a problem to flight because the clutch will definitely be fully engaged in flight. Curtis commented that it flew much better, seeming to have much more power and being able to respond better. We were able to put the helicopter into a much more stable autonomous hover at this time and were able to command it to fly to GPS waypoints that we designated by clicking on the moving map display on the ground station. After the helicopter had flown for some time

autonomously, we experienced a communication failure from the AFCS data. This failure was not as problematic as the communication failures with the test pilot transmitter as the pilot never lost control of the vehicle. The ground station just couldn't get data from it or send it any commands. Curtis then proceeded to land the helicopter manually and when he started to throttle it down after it had landed it started making a metal on metal scraping sound. In order to diagnose the noise, we again pulled the engine from the helicopter to see if it was functioning properly. Our initial thought was that we might have to swap the engine with the G23 if the G26 had gone bad, but upon inspection Curtis decided that the engine was not the problem. Examining the clutch he found that the bearing inside the clutch bell, which is responsible for the clutches free rotation within the bell at lower motor speeds, had locked up. This explained why the rotor head was pulling so hard as start up and it also displayed marks indicating metal on metal rubbing between the clutch shaft and the inside of the bearing which would explain the noise that we had heard. Knowing that we could still fly successfully with the clutch in this condition, Curtis replaced the clutch in the helicopter while trying to alleviate as much pressure as possible on that bearing. We tried a series of flights after that and had great success in autonomous flight to various waypoints and decided to try out the obstacle avoidance system. I had Curtis put the helicopter in an autonomous hover facing the side of the flight lab building about 40 feet away from it. I then had him flip the switch to activate the obstacle avoidance system as I selected a waypoint on the other side of the building with the ground station. The system did demonstrate reactions commensurate with the avoidance algorithm, but it did it too far from the wall to be able to actually detect the wall. We attempted this test a few times with similar results.

Looking at the sensor readings from the ground station I found that the sensors were giving false readings which is why the helicopter was reacting before it reached the wall. With this realization we decided not to fly any more that day until I could figure out what was wrong with the sensors or replace them.

After Curtis had left, I took the helicopter inside to the office where I had done the initial tests with the sensors to evaluate them and found that they were all functioning properly inside. After thinking about the problem more I remembered that I had encountered some false readings with the sensors on the push tests I performed when the sensor seemed to be pointing into the wind. Discussing the problem with the manufacturers of the sensors they recommended some kind of cone to go around the sensor as a possible solution. Experimenting with various types of shielding along that line I achieved little success in eliminating the sensor errors. This closer inspection actually showed me that wind didn't seem to be as big a factor as I had thought. It seemed that some of the sensors would just work fine sometimes and other times they wouldn't. The sensors that seemed to be most susceptible to errors were the ones located on the ends of the array. Discussing the problem with Dr. Langari, we decided that I should modify the program to only use the middle three sensors and see if the program would work better that way.

Once the program modifications were complete I called Curtis to come out once again for another flight. During this flight test, the data communication from the AFCS kept dropping out while Curtis was throttling the helicopter up to the point that it could not get off the ground without losing communication. Curtis and I spent some time looking over parts of the helicopter trying to theorize why this problem was occurring but

could not come up with a viable solution. Without being able to communicate with the AFCS via the ground station, it was not productive to do any more flights that day. We did some sensor checks with the helicopter on the ground and found that they were still not consistently responding properly. At this point, I let Curtis leave early and called Dennis at Rotomotion to discuss the issues I was having with data communication. He said they had experienced that problem before and explained that the current board to which they had the communication module mounted was fairly susceptible to EMI and that the EMI would overload the communication circuit causing it to stop working. The EMI does not actually damage the circuit and therefore the communication can be reestablished by restarting the AFCS. These symptoms were identical to the ones we were seeing in our system. With EMI appearing to be the problem with the communication I thought back other problems we had been having with the helicopter and decided that the only major change in the helicopter operation that we had seen since we started having this problem was the locking of the bearing in the clutch bell. Deciding that this could be a possible source of EMI as it involved metal on metal rubbing which could lead to sparking, I called Larry Bergen at Bergen RC and ordered a new bearing for the clutch bell. When the new bearing arrived, I removed the clutch and bell from the helicopter once again and removed the bearing from the clutch bell by heating the outer surface of the bell so that it would expand until I could easily tap it out with a hammer and screwdriver, which was the method prescribed to me by Larry Bergen. Once old bearing was out, I checked it and it appeared to be able to rotate once it was free from the bell. I put the new bearing in the bell by putting a piece of metal with a flat surface against it and pounding it into the location of the old bearing. This bearing

demonstrated similar problems to the previous one in that it was also unable to rotate. I removed this bearing from the bell and analyzed the bell. Upon inspection of the bearing location in the bell I found that the shaft from the top of the bell protruded slightly down into the bearing location with a shaft that was of similar diameter to that of the inner sleeve of the bearing. This means that if the bearing is pressed fully into the bell, the protruding piece would be in contact with the inner sleeve of the bearing and prevent it from rotating. Discussing this problem with Curtis at a later time he said that the pinion gear shaft, which is the top portion of the clutch bell, is removable and must have been put in too far when Bergen reworked the clutch bell as we did not see this problem until we received it back from them. As a possible solution to this problem, I press fitted the bearing back into the bell in a slow and careful manner to monitor the depth of the bearing into the bell and did not press it in all the way so that it did not come in contact with the shaft of the bell which allowed the bearing to rotate freely.

After performing a strap down test to ensure that this solution fixed the data communication problem, I called Curtis back out for our final test flight. The helicopter flew almost flawlessly this flight and was used in autonomous mode for a total of almost two hours during these tests. We again tried to verify the functionality of the obstacle avoidance system by setting up the helicopter in an autonomous hover at a distance of about 40 feet from the wall of the flight lab and sending it to a waypoint on the other side of the flight lab. The system did not seem to respond to the wall after repeated attempts so I started try to diagnose the problem by requesting information from the obstacle avoidance microcontroller via the ground station. The microcontroller did not respond to repeated inquiries while in flight. In attempt to diagnose this problem I set up the

helicopter off the ground a few feet to ensure that the sensors did not see the ground, which would interrupt the communication loop that would send the information, and found that the communication worked fine. This being the case I started the obstacle avoidance program while it was on the ground and monitored its activity as Curtis flew it up off of the ground. Communication with the microcontroller seemed to drop out after the helicopter rose off of the ground about 10 to 20 feet. We also seemed to have communication dropouts from the AFCS when it was lifting off of the ground. The difference between these communication dropouts and those that we experienced before was that communication could be reestablished by repeatedly pressing the reconnect button on the ground station every time the communication stopped. Once the helicopter was up in the air at a regular hover, the data communication stopped trying to drop out and we could resume normal flight tests. The communication dropouts from the obstacle avoidance microcontroller appeared to be intermittent as well and we did get it to function during one approach to the wall in autonomous mode. We did see the helicopter exhibit behavior indicating that it entered the avoidance loop by pointing the sensor turret to the left to check for obstacles in its desired new path as it approached the wall. It performed this action right as the helicopter was within a reasonable sensor range from the wall, but at the exact time that the system reacted Curtis had to take control of the helicopter because it was losing altitude and he feared that it might crash into the ground. We repeated a few more attempts at flying toward the wall but continually encountered the problem of the helicopter either gaining altitude or losing altitude to the point that it could not see the wall because it was too high above the wall of the flight lab or it had to be manually taken over to prevent it from flying into the ground. We theorized that the

source of this problem could either be the downwash off of the building from the wind coming from the other side or the loss of GPS accuracy because loss of satellites due to the increasingly close proximity of the wall of the building which could block the signal. After flying for a few hours we had to stop as there was a front coming through College Station that afternoon and the wind speed kicked up dramatically and it had begun to rain.

Knowing that this was the last flight as Curtis had already done one more flight than he agreed to initially and said he just did not have the time to do any more, I decided to perform a few more tests in a strap down scenario to trouble shoot the microcontroller and sensor consistency problems. Dave Lund and I discussed the best way to diagnose this problem and we decided to strap down the shipping pallet we had been using for our other strap down tests to the top of a table and placed additional weights on the table to ensure its stability as the helicopter was running. We started by running the obstacle avoidance system without the engine running to work just with the sensors. Since we were running this test inside the hanger, we hoped that that would eliminate any problems we might have from wind interfering with the sensors. In order to test the effects of wind in a more structured manner, Dave took a shop vacuum/blower and blew high velocity air at individual sensors from a variety of angles while I monitored the sensor readings. We did not see any variations in the sensor readings relating to the wind velocity or direction in this experiment. The next test was to check the operation of the microcontroller as the helicopter engine was running. Again we saw communication dropouts as the engine speed increased and the clutch and main rotor began to engage. To diagnose this problem I connected the serial cable to microcontroller to bypass the need for communication through the wireless access point to eliminate that variable. I then covered all of the

electronic components of the helicopter with aluminum foil and tried the test again. The microcontroller did not lose communication during this engine run up, verifying that the communication dropout was an EMI problem similar to that which we saw with the AFCS in previous tests. The communication indicator lights on the microcontroller flashed rapidly when it was communicating with the AFCS and ground station and stayed on but stopped flashing when the communication dropped out indicating that there was still power to the controller but it was unable to communicate. To diagnose which portion of the electronics was responsible for the EMI vulnerability I systematically removed one portion of foil at a time and then monitored the microcontroller communication through a range of engine throttle angles as it was running. I found that I covered the long bundle of wires from the receiver box to the servos in the rear of the plane and the collection of wires entering the access point/router/microcontroller box and left the rest of the components bare, the microcontroller would be able to function through the full range of engine RPMs. Testing the sensors more with the engine running I found the far left sensor to be completely broken and the other sensors wandered as time passed after they were turned on. This showed that these particular sensors would not be a reliable option for implementation in this system.

CONCLUSION AND RECOMMENDATIONS

The Texas A&M Autonomous Helicopter System is now functioning in a reliable and consistent manner in which it can maintain an autonomous hover and fly to GPS waypoints as they are assigned from the ground station. The data communication does still demonstrate problems during take off, but communication can be reestablished by pressing the reconnect button on the ground station as soon as the communication drops out. The obstacle avoidance system has performed successfully in simulations in which the algorithm is identical to that used in the real world version and the helicopter simulation program written by Rotomotion interfaces with the obstacle avoidance microcontroller using identical communication packets to those used the real flight AFCS communication. Obstacle avoidance ground tests performed with the helicopter on a cart also proved to be successful. In this test all of the electronics on the helicopter are powered and functioning while the engine remains off and the operator pushes the helicopter to the GPS waypoints indicated by the ground station as a result of obstacle avoidance commands dictated by readings from the sensors. In actual flight tests, the helicopter did again follow the proper commands while in flight moving the sensor turret to the proper directions and flying to GPS points that would lead it around an obstacle. The problem with these flights was that the sensors were giving false readings which caused the helicopter to react even though there were no objects within its sensing range. Reliability of all electrical/communication systems has been improved by adjusting the relative position of various components and adding appropriate EMI shielding based on test results.

For future experiments I would make the following recommendations based on my experience with this project:

- Get a pilot on contract in some manner that they have an obligation to show up for a certain amount of time for at least once a week if needed. Problems with accommodating and waiting for pilot's schedules for test flights on this project made it take at least twice as long to complete it.
- Replace the current Senscomp ultrasonic sensors with sturdier more reliable sensors. The new sensors should be selected based on their ability to detect the specific types of surfaces that might be encountered by the helicopter as obstacles. Extended sensor range and faster sensor update times would allow for higher maximum transit speeds between obstacles. The sensors also need to be rugged enough to handle the vibration inherent in an airborne helicopter system and considerations must be taken to deal the EMI created by this system.
- Replace the current helicopter with a better built, more reliable, more powerful helicopter with a bigger payload capacity. Many of the problems encountered in developing this system arose from fixing or overcoming shortcomings inherent in the design and manufacture of this helicopter. Rotomotion, who originally started their work with Bergen Observers, later started to make their own body frames for this helicopter out of aluminum to increase its rigidity and eliminate the electrical potential difference problems created by the G10 frame. They are now starting to get away from Observers altogether. The belt drive tail rotor system is no longer even produced by Bergen on their new Observers and has been replaced by a solid

drive shaft going through the tail boom, presumably because of similar issues to the ones we experienced.

- A possible doctoral dissertation for an individual would be to develop A&M's own AFCS style controller with an IMU, GPS and onboard processor running some kind of control scheme.
- Many autonomous helicopter projects seem to be incorporating some kind of vision system into their helicopter, either for an obstacle avoidance alternative or for surveillance purposes. For instance, the AUVSI International Aerial Robotics Competition requires reconnaissance data to be collected by a helicopter and relayed to the ground station.

REFERENCES

[1] Association for Unmanned Vehicle Systems International, *International Aerial Robotics Competition: Launch Point for Aerial Robotics*. http://avdil.gtri.gatech.edu/AUVS/IARCLaunchPoint.html Accessed December 2005

[2] Association for Unmanned Vehicle Systems International, *International Aerial Robotics Competition: Rules*. http://avdil.gtri.gatech.edu/AUVS/CurrentIARC/200xCollegiateRules.html

[3] Association for Unmanned Vehicle Systems International, *Unmanned Systems Online: News*. http://www.auvsi.org/news/ Accessed December 2005

[4] D. D'Annunzio and T. Hudson, *UAV User's Guide Rotomotion, LLC.* Charleston, SC: Rotomotion LLC. August 26, 2004.

[5] Defense Advanced Research Projects Agency, *DARPA and Army Select Unmanned Combat Armed Rotorcraft Contractors*. May 28, 2002. http://www.darpa.mil/body/news/2002/ucar.pdf

[6] Department of Defense, Unmanned *Aircraft Systems Roadmap 2005-2030*. August 4, 2005. http://www.acq.osd.mil/usd/Roadmap%20Final2.pdf

[7] T. Hudson, *API User's Guide Rotomotion, LLC.* Charleston, SC: Rotomotion LLC. February 9, 2005.

[8] Intec Automation, Inc. *Wildfire 32-bit Microcontroller User Manual, Version 1.1.1*. Victoria BC, Canada. January 17, 2005.

[9] Intec Automation, Inc. *Wildfire Programmer Reference Manual, Version 1.1*. Victoria BC, Canada. December 30, 2004.

[10] A. Proctor, B. Gwin, S. Kannan, A. Koller, H. Christopherson, E. Johnson. *Ongoing Development of an Autonomous Aerial Reconnaisance System at Georgia Tech*. 2004. Paper presented for AUVSI IARC Competition http://controls.ae.gatech.edu/gtar/iarcpapers/git2004.pdf

[11] Rotomotion, LLC. *Diagram of Flight Controller Integration* http://rotomotion.com/prd_UAV_CTLR.html Accessed May 2005

APPENDIX A

SIMULATION OBSTACLE AVOIDANCE CODE

```
#include <wf5282.h>        // GPTPort, ...
#include <stdio.h>         // printf
#include <stdlib.h>                    // strtod
#include <opentcp.h>       // NIF and all other OpenTcp


//------------- Implementation ------------------------------------------
int udp_eventlistener(char , char , int , short , short , short );
char udp_socket;
UINT32 *remote_ip;
char connected, desired, intransit;
UINT8  genericsign, genericsign2, random;
UINT32 genericint, genericint2;
UINT32 genericdec, genericdec2;

//UINT8  vel_nsign, vel_esign, vel_dsign;
//UINT8  numsat, vdop;
//UINT32 vel_nint, vel_eint, vel_dint;
//UINT32 vel_ndec, vel_edec, vel_ddec;
UINT8  vxsign, vysign, vzsign, thetasign, psisign;
UINT8  xsign, ysign, zsign;
UINT32 vxint, vyint, vzint, thetaint, psiint, axint, ayint;
UINT32 xint, yint, zint;
UINT32 vxdec, vydec, vzdec, thetadec, psidec;
UINT32 xdec, ydec, zdec;
UINT8 bytbuf[8];

UINT8  desirednsign, desiredesign, desireddsign, desiredhdgsign, looksign;
UINT32 desirednint, desiredeint, desireddint, desiredhdgint, lookint;
UINT32 desiredndec, desirededec, desiredddec, desiredhdgdec, lookdec;
UINT8  tempnsign, tempesign, temphdgsign;
UINT32 tempnint, tempeint, temphdgint;
UINT32 tempndec, tempedec, temphdgdec;

UINT32 *secip;
short secport;

                    // TYPE: Main
int                  // return error code (not used)
   main( void )          // main function called from crt0.s -> from dBUG
{
   UINT8  wallside, inloop;
   UINT32 look;
         UINT8 *buf_ptr;
   UINT8 *buf;
   UINT16 count;
```

```
   UINT8 chan;
   unsigned short sensor[5], distance, diff, diffprev;
   UINT8  orignsign, origesign, origdsign, orighdgsign, tempnsign, tempesign,
deltalooksign;
        UINT32 orignint, origeint, origdint, orighdgint, tempnint, tempeint, deltalookint;
        UINT32 origndec, origedec, origddec, orighdgdec, tempndec, tempedec,
deltalookdec;

        UINT8 lookmode, obstacle;
        UINT32 pitchint;
        UINT32 pitchdec;
        UINT8  pitchsign;
        UINT32 psiwantint;
        UINT32 psiwantdec;
        UINT8  psiwantsign;
        UINT32 n;
        UINT32 xsqrint, ysqrint, deltanint, deltaeint, prop, propint, newnint, neweint;
        UINT32 xsqrdec, ysqrdec, deltandec, deltaedec, propdec, newndec, newedec;
        UINT8  xsqrsign, ysqrsign, deltansign, deltaesign, propsign, newnsign, newesign;
        random = 0;
        UINT32 xsin50, xcos50, xsin25, xcos25;
        UINT32 deltax, deltay;

        wdt_timeout_set( 0x0FA0 );
   //Initialize TCP/IP subsystem
   tcpip_init(0);
   //Open a UDP socket for communication
   udp_socket = udp_getsocket(0 , udp_eventlistener, 0);
   if( udp_socket == -1 ) {
        printf("DNS: No free UDP sockets!! \r\n");
        return; }
   //Bind socket to port and open for communication
   if( udp_open( udp_socket, 2002 ) >= 0 ) {
        puts( "Opened UDP socket on local port 2002\n" ); }
   //Send Open command to AFCS in loop until acknowledge packet received
   remote_ip = make_ip( 10, 255, 0, 5 );

        connected = 1;
        while(connected==1){
                wdt_feed();
           buf = buf_ptr = OTCP_TXBUF + UDP_APP_OFFSET;
           *buf_ptr++ = 1;  //unimportant time padding
           *buf_ptr++ = 2;
           *buf_ptr++ = 3;
           *buf_ptr++ = 4;
           *buf_ptr++ = 9;
```

```
            *buf_ptr++ = 6;
            *buf_ptr++ = 7;
            *buf_ptr++ = 8;
            *buf_ptr++ = 0;  //Identifies as packet Id "Open"
            *buf_ptr++ = 0;
            *buf_ptr++ = 0;
            *buf_ptr++ = 1;
            *buf_ptr++ = 0;  //extra byte needed for this controller
            udp_send( udp_socket, remote_ip, 2002, buf,
NETWORK_TX_BUFFER_SIZE - UDP_APP_OFFSET, 13 );
            tcp_tick();
         }
         ad_init(                // Initialize the AD converter
               AD_SAMPLE_16TICS,      // 16 Ticks (8 microseconds sampling time)
               AD_8CHAN_8SCAN );     // Get average of 8 most recent scans

         count=0;

   while(1){
         wdt_feed();
//       count += 1;
//       printf("count = %lu ",count);
         tcp_tick();

//          convertxytone(1,0,0,0,0,0);
//          printf("n = %u %lu.%08lu, e = %u %lu.%08lu\n",genericsign, genericint,
genericdec,
//                genericsign2, genericint2, genericdec2);
//       vxsign = 0;
//       vxint = 1;
//       vxdec = 0;
//       vysign = 0;
//       vyint = 1;
//       vydec = 0;
//       vzsign = 0;
//       vzint = 0;
//       vzdec = 0;

//       printf("vx %d %2lu.%08lu, vy %d %2lu.%08lu, vz %d %2lu.%08lu\n",
//                vxsign, vxint, vxdec,
//                vysign, vyint, vydec, vzsign, vzint, vzdec);
         //when faster than 1 m/s point platform in that direction
         look = 0;
         lookint = 0;
         lookdec = 0;
         looksign = 0;
```

```
        if (((vxint+vyint)>0)||((vxdec+vydec)>100000000)){
                decintadd(vxint, vxdec, 0, vyint, vydec, 0);
//              printf("generic %lu.%08lu\n",genericint, genericdec);
                look = ((vyint*1000000+(vydec/100))/
                        (genericint*1000+(genericdec/100000)))*(1571);
                if (vxsign == 0){
                        if (vysign == 0){
                                looksign = 0;
                        }
                        else {
                                looksign = 1;
                        }
                }
                else {
                                look = 31415926-look;
                        if (vysign == 0){
                                looksign = 0;
                        }
                        else {
                                looksign = 1;
                        }
                }
                  lookint = look/1000000;
                  lookdec = (look-((look/1000000)*1000000))*100;
                        if (thetasign == 0){
                                lookat(0, thetaint, thetadec, 1, lookint, lookdec, looksign);
                        }
                        else{
                                lookat(0, thetaint, thetadec, 0, lookint, lookdec, looksign);
                        }
//                  lookat(0, 0, 0, 0, 0, 0, 0);
                        if ((lookint>0)&&(lookdec>57079633)){
                                lookint = 1;
                                lookdec = 57079633;
                        }
        }
      else if (thetasign == 0){
                        lookat(0, thetaint, thetadec, 1, 0,0,0);
                        lookint = 0;
                        lookdec = 0;
                        looksign = 0;
                }
                else{
                        lookat(0, thetaint, thetadec, 0, 0,0,0);
                        lookint = 0;
                        lookdec = 0;
```

```
                looksign = 0;
            }
        //read sensors in cm
        for( chan = 0; chan < 5; chan++ ) {
           sensor[chan] =  sensorinput(chan);
        }
        buf = buf_ptr = OTCP_TXBUF + UDP_APP_OFFSET;
            *buf_ptr++ = 1;  //unimportant time padding
            *buf_ptr++ = 2;
            *buf_ptr++ = 3;
            *buf_ptr++ = 4;
            *buf_ptr++ = 9;
            *buf_ptr++ = 6;
            *buf_ptr++ = 7;
            *buf_ptr++ = 8;
            *buf_ptr++ = 0;  //Identifies as packet Id "afcs"
            *buf_ptr++ = 0;
            *buf_ptr++ = 0;
            *buf_ptr++ = 15;
            *buf_ptr++ = 0;
            *buf_ptr++ = sensor[0]/10;
            *buf_ptr++ = 0;
            *buf_ptr++ = sensor[1]/10;
            *buf_ptr++ = 0;
            *buf_ptr++ = sensor[2]/10;
            *buf_ptr++ = 0;
            *buf_ptr++ = sensor[3]/10;
            *buf_ptr++ = 0;
            *buf_ptr++ = sensor[4]/10;
            *buf_ptr++ = 0;
            *buf_ptr++ = 0;
            *buf_ptr++ = 0;
            udp_send( udp_socket, secip, secport, buf, NETWORK_TX_BUFFER_SIZE
                    - UDP_APP_OFFSET, 24 );
                tcp_tick();
            printf("Sensor Distances ");
            printf("0: %u, 1: %u, 2: %u, 3: %u, 4: %u\n", sensor[0], sensor[1],
                sensor[2], sensor[3], sensor[4]);
            if ((sensor[2] < 800)&&(sensor[2]>35)){ //obstacle detected by center sensor
//                  if ((sensor[1]<1200)&&(sensor[3]<1200)){
                        wdt_feed();
                        //keep original destination in memory to reinput after obstacle is
passed
                        tcp_tick();
                        orignint = desirednint;
                        origndec = desiredndec;
```

```
                    orignsign = desirednsign;
                    origeint = desiredeint;
                    origedec = desirededec;
                    origesign = desiredesign;
                    origdint = desireddint;
                    origddec = desiredddec;
                    origdsign = desireddsign;
                    orighdgint = desiredhdgint;
                    orighdgdec = desiredhdgdec;
                    orighdgsign = desiredhdgsign;
                    //stop 7 meters from obstacle, keep sending until recognized by
AFCS
                        sensor[2] =  sensorinput(2);
                    distance = sensor[2]-400;
                    if (xsign == 0){
                        decintadd(orignint, origndec, orignsign, xint, xdec, 1);
                    }
                    else{
                        decintadd(orignint, origndec, orignsign, xint, xdec, 0);
                    }
                    deltanint = genericint;
                    deltandec = genericdec;
                    deltansign = genericsign;
                    decintmult(deltanint, deltandec, 0, deltanint, deltandec, 0);
                        xsqrint = genericint;
                        xsqrdec = genericdec;
                    if (ysign == 0){
                        decintadd(origeint, origedec, origesign, yint, ydec, 1);
                    }
                    else{
                        decintadd(origeint, origedec, origesign, yint, ydec, 0);
                    }
                    deltaeint = genericint;
                    deltaedec = genericdec;
                    deltaesign = genericsign;
                    decintmult(deltaeint, deltaedec, 0, deltaeint, deltaedec, 0);
                    ysqrint = genericint;
                        ysqrdec = genericdec;
                        decintadd(xsqrint, xsqrdec, 0, ysqrint, ysqrdec, 0);
                        squareroot(genericint, genericdec);
                        prop =
(distance*1000000)/((genericint*100)+(genericdec/1000000));
                        propint = prop/1000000;
                        propdec = (prop - (propint*1000000))*100;
                        decintmult(deltanint, deltandec, 0, propint, propdec, 0);
```

```
                                decintadd(genericint, genericdec, deltansign, xint, xdec,
xsign);
                                newnint = genericint;
                                newndec = genericdec;
                                newnsign = genericsign;
                                decintmult(deltaeint, deltaedec, 0, propint, propdec, 0);
                                decintadd(genericint, genericdec, deltaesign, yint, ydec,
ysign);
                                neweint = genericint;
                                newedec = genericdec;
                                newesign = genericsign;

        flyto(16,1,2,newnint,newndec,newnsign,neweint,newedec,newesign,
                                origdint,origddec,origdsign,lookint,lookdec,looksign);
//                              flyto(16,4,2,distance/100,(distance-
((distance/100)*100))*1000000,0,0,0,0,lookint,lookdec,looksign);
                                intransit = 1;
                                printf("stop\n");
                                while(intransit==1){
                                        wdt_feed();
                                        tcp_tick();
//                                      printf("ax=%lu, ay=%lu, vx=%lu, vy=%lu,
desirednint=%lu,xint=%lu, desiredeint=%lu, yint=%lu\n",axint, ayint,vxint, vyint,
desirednint, xint, desiredeint, yint);
//                                      printf("look = %u %lu.%08lu, psi
= %u %lu.%08lu\n"looksign, lookint, lookdec, psisign, psiint, psidec);

                                        if ((axint==0)&&(ayint==0)){
                                                if ((vxint==0)&&(vyint==0)){
                                                        if (abs(desirednint-xint)<3){
                                                                if(abs(desiredeint-yint)<3){
//                                                                      if ((psiint-
lookint)==0){
//                                                                              if (((psidec-
lookdec)/1000000)==0){

        intransit = 0;
//                                                                      }
//                                                              }
                                                        }
                                                }
                                        }
                                }
                        }
```

```
                                printf("wall\n");
//
//                      for( chan = 0; chan < 5; chan++ ) {
//                         sensor[chan] =  sensorinput(chan);
//                      }
//                                //determine relative plane of the wall
//                                deltalookint = lookint;
//                                deltalookdec = lookdec;
//                                deltalooksign = looksign;
//                                if (sensor[1]!=sensor[3]){
//                                        if (sensor[1]>sensor[3]){
//                                                while ((sensor[1] >
sensor[3])&&(lookint<2)){
//                                                        for (n=0;n<15000000;n++){
//                                                                wdt_feed();
//                                                        }
//                                                        wdt_feed();
//                                                        diffprev = sensor[1]-sensor[3];
//                                                        sensor[1] =  sensorinput(1);
//                                                        sensor[3] =  sensorinput(3);
//                                                                printf("1  %u  >  3  %u
",sensor[1], sensor[3]);
//                                                                printf("
look %lu.%08lu\n",lookint,lookdec);
//                                                        decintadd(lookint, lookdec, looksign,
0, 10000000, 0);
//                                                        lookint = genericint;
//                                                        lookdec = genericdec;
//                                                        looksign = genericsign;
//                                                        lookat(0, 0, 0, 0, lookint, lookdec,
looksign);
//                                                }
//                                                diff=sensor[3]-sensor[1];
//                                                if (diff>diffprev){
//                                                        decintadd(lookint, lookdec, looksign,
0, 10000000, 1);
//                                                        lookint = genericint;
//                                                        lookdec = genericdec;
//                                                        looksign = genericsign;
//                                                        for (n=0;n<15000000;n++){
//                                                                wdt_feed();
//                                                        }
//                                                        lookat(0, 0, 0, 0, lookint, lookdec,
looksign);
//                                                }
//                                        }
```

```
//                                      else {
//                                              while ((sensor[1] <
sensor[3])&&(lookint<2)){
//                                                      for (n=0;n<15000000;n++){
//                                                              wdt_feed();
//                                                      }
//                                                      wdt_feed();
//                                                      diffprev = sensor[3]-sensor[1];
//                                                      sensor[1] =  sensorinput(1);
//                                                      sensor[3] =  sensorinput(2);
//                                                              printf("1  %u  <  3  %u
",sensor[1], sensor[3]);
//                                                              printf("
look %lu.%08lu\n",lookint,lookdec);
//                                                      decintadd(lookint, lookdec, looksign,
0, 10000000, 1);
//                                                      lookint = genericint;
//                                                      lookdec = genericdec;
//                                                      looksign = genericsign;
//                                                      lookat(0, 0, 0, 0, lookint, lookdec,
looksign);
//                                              }
//                                              diff=sensor[1]-sensor[3];
//                                              if (diff>diffprev){
//                                                      decintadd(lookint, lookdec, looksign,
0, 10000000, 0);
//                                                      lookint = genericint;
//                                                      lookdec = genericdec;
//                                                      looksign = genericsign;
//                                                      for (n=0;n<15000000;n++){
//                                                              wdt_feed();
//                                                      }
//                                                      lookat(0, 0, 0, 0, lookint, lookdec,
looksign);
//                                              }
//                                      }
//
//                              }
//                              for (n=0;n<10000000;n++){
//                                      wdt_feed();
//                              }
//                              printf("done");
//                              for (n=0;n<100000000;n++){
//                                      wdt_feed();
//                              }
//                              if (deltalooksign == 0){
```

```
//                                      decintadd(lookint, lookdec, looksign, deltalookint,
deltalookdec, 1);
//                              }
//                              else{
//                                      decintadd(lookint, lookdec, looksign, deltalookint,
deltalookdec, 0);
//                              }
//                              deltalooksign = genericsign;
//                              printf("look = %lu %lu.%08lu\n",looksign, lookint,
lookdec);
//                              if (deltalooksign==1){
//                                      if (looksign == 0){
//                                              decintadd(1, 57079633, 0, lookint, lookdec,
0);
//                                              decintadd(psiint, psidec, psisign, genericint,
genericdec, 0);
//                                      }
//                                      else{
//                                              decintadd(1, 57079633, 0, lookint, lookdec,
1);
//                                              decintadd(psiint, psidec, psisign, genericint,
genericdec, 0);
//                              }
//                              }
//                              else{
//                                      if (looksign == 1){
//                                              decintadd(1, 57079633, 0, lookint, lookdec,
0);
//                                              decintadd(psiint, psidec, psisign, genericint,
genericdec, 1);
//                                      }
//                                      else{
//                                              decintadd(1, 57079633, 0, lookint, lookdec,
1);
//                                              decintadd(psiint, psidec, psisign, genericint,
genericdec, 1);
//                              }
//                              }
//                              printf("hdg = %lu.%08lu\n",genericint, genericdec);
//                              printf("psi = %u %lu.%08lu\n",psisign, psiint, psidec);
//                              psiwantint = genericint;
//                              psiwantdec = genericdec;
//                              psiwantsign = genericsign;
//                              if (psiwantint>2){
//                                      if ((psiwantdec>14159265)||(psiwantint>3)){
//                                              if (psiwantsign == 1){
```

```
//                                              psiwantsign = 0;
//                                      }
//                                      else{
//                                              psiwantsign = 1;
//                                      }
//                                      decintadd(6, 28318531, 0, psiwantint,
psiwantdec, 1);
//                                      psiwantint = genericint;
//                                      psiwantdec = genericdec;
//                              }
//                      }
//                      printf("psiwant = %u %lu.%08lu\n",psiwantsign,
psiwantint, psiwantdec);

                        wallside = 2;
                        while (wallside == 2){
                                wdt_feed();
                                lookint = 0;
                                lookdec = 87266463;
                                looksign = 1;
                                lookat(0, thetaint, thetadec, thetasign, 0, 87266463,
1);
                                for (n=0;n<3000;n++){
                                        printf("feed\n");
                                        wdt_feed();
                                }
                                sensor[0]=sensorinput(0);
                                if (sensor[0] > 1200){
                                        wallside = 4;
                                        flyto(16,1,2,xint,xdec,xsign,yint,
                                                ydec,ysign,zint,zdec,zsign,
                                                1, 57079633, 1);
                                        intransit = 1;
                                        while(intransit==1){
                                                wdt_feed();
                                                tcp_tick();
                                                if (desiredhdgsign == psisign){
                                                        if (desiredhdgint == psiint){
                                                                if
((desiredhdgdec/1000000)==(psidec/1000000)){
                                                                        intransit = 0;
                                                                }
                                                        }
                                                }
                                        }
                                        lookint = 0;
```

```
                                        lookdec = 87266463;
                                        looksign = 0;
                                        lookat(0, thetaint, thetadec, thetasign, 0,
87266463, 0);
                                        for (n=0;n<6000;n++){
                                                printf("feed\n");
                                                wdt_feed();
                                        }
                                }
                                else{
                                        lookint = 0;
                                        lookdec = 87266463;
                                        looksign = 0;
                                        lookat(0, thetaint, thetadec, thetasign, 0,
87266463, 0);
                                        for (n=0;n<3000;n++){
                                                printf("feed\n");
                                                wdt_feed();
                                        }
                                        sensor[4]=sensorinput(4);
                                        if (sensor[4] > 1200){
                                                wallside = 0;
                                                flyto(16,1,2,xint,xdec,xsign,yint,
                                                        ydec,ysign,zint,zdec,zsign,
                                                        1, 57079633, 0);
                                                intransit = 1;
                                                while(intransit==1){
                                                        wdt_feed();
                                                        tcp_tick();
                                                        if (desiredhdgsign ==
psisign){
                                                                if (desiredhdgint ==
psiint){
                                                                        if
((desiredhdgdec/1000000)==(psidec/1000000)){

        intransit = 0;
                                                                        }
                                                                }
                                                        }
                                                }
                                                lookint = 0;
                                                lookdec = 87266463;
                                                looksign = 1;
                                                lookat(0, thetaint, thetadec, thetasign,
0, 87266463, 1);
```

```
                                                for (n=0;n<6000;n++){
                                                        printf("feed\n");
                                                        wdt_feed();
                                                }
                                        }
                                }
                                if (wallside == 2){
                                        reltoabs(5,0,1,0,0,0);
                                        flyto(16,1,2,genericint, genericdec,
genericsign,
                                                genericint2, genericdec2,
genericsign2,
                                                origdint, origddec, origdsign, 0,0,0);
//                                      flyto(16,4,2,5,0,1,0,0,0,0,0,0,0,0,0);
                                        intransit = 1;
                                        printf("back up\n");
                                        while(intransit==1){
                                                wdt_feed();
                                                tcp_tick();
                //                              printf("ax=%lu, ay=%lu, vx=%lu,
vy=%lu, desirednint=%lu,xint=%lu, desiredeint=%lu, yint=%lu\n",axint, ayint,vxint,
vyint, desirednint, xint, desiredeint, yint);
                                                if ((axint==0)&&(ayint==0)){
                                                        if
((vxint==0)&&(vyint==0)){
                                                                if (abs(desirednint-
xint)<3){

        if(abs(desiredeint-yint)<3){

        intransit=0;
                                                                        }
                                                                }
                                                        }
                                                }
                                        }
                                }

                        tcp_tick();
                                for( chan = 0; chan < 5; chan++ ) {
                        sensor[chan] =  sensorinput(chan);
                }
                printf("Sensor Distances ");
                     printf("0: %u, 1: %u, 2: %u, 3: %u, 4: %u\n", sensor[0],
sensor[1],
```

```
                                sensor[2], sensor[3], sensor[4]);
                                obstacle = 1;
                                if (wallside == 4){
                                        printf("1");
                                        while(obstacle == 1){
                                                wdt_feed();
                                                printf("2");
                                                obstacle = 0;
                                                if (sensor[0]>1100){
                                                        printf("3");
                                                        if ((sensor[4]<1100)){
                                                                obstacle = 1;
                                                                for( chan = 0; chan < 5;
chan++ ) {
                                                        sensor[chan] =  sensorinput(chan);
                                                    }
                                                    printf("Sensor Distances ");
                                                         printf("0: %u, 1: %u, 2: %u, 3: %u,
4: %u\n", sensor[0], sensor[1],
                                                                sensor[2], sensor[3],
sensor[4]);
                                                     if (sensor[4]<600){
//                                                          if ((sensor[3]<600)){
                                                                    if (looksign == 1){

        reltoabs(5,0,0,7-(sensor[4]/100),0,0);

        flyto(16,1,2,genericint, genericdec, genericsign,

        genericint2, genericdec2, genericsign2,

        origdint, origddec, origdsign, 0,0,0);
//
        flyto(16,4,2,5,0,0,7-(sensor[4]/100),0,0,0,0,0,0,0,0);
                                                                            tcp_tick();
                                                                }
                                                                else{


        reltoabs(5,0,0,7-(sensor[4]/100),0,1);

        flyto(16,1,2,genericint, genericdec, genericsign,

        genericint2, genericdec2, genericsign2,

        origdint, origddec, origdsign, 0,0,0);
```

```
//
        flyto(16,4,2,5,0,0,7-(sensor[4]/100),0,1,0,0,0,0,0,0);
                                                                tcp_tick();
                                                        }
//                                                  }
                                                }
                                                else if (sensor[4]>600){
//                                                  if ((sensor[3]>600)){
                                                        if (looksign == 1){

        reltoabs(5,0,0,(sensor[4]/100)-5,0,0);

        flyto(16,1,2,genericint, genericdec, genericsign,

        genericint2, genericdec2, genericsign2,

        origdint, origddec, origdsign, 0,0,0);
//
        flyto(16,4,2,5,0,0,(sensor[4]/100)-5,0,1,0,0,0,0,0,0);
                                                                tcp_tick();
                                                        }
                                                        else{

        reltoabs(5,0,0,(sensor[4]/100)-5,0,1);

        flyto(16,1,2,genericint, genericdec, genericsign,

        genericint2, genericdec2, genericsign2,

        origdint, origddec, origdsign, 0,0,0);
//
        flyto(16,4,2,5,0,0,(sensor[4]/100)-5,0,0,0,0,0,0,0,0);
                                                                tcp_tick();
                                                        }
//                                                  }
                                                }
                                                else{
                                                    reltoabs(5,0,0,0,0,0);
                                                            flyto(16,1,3,genericint,
genericdec, genericsign,
                                                                genericint2,
genericdec2, genericsign2,
                                                                origdint,
origddec, origdsign, 0,0,0);
//
        flyto(16,4,3,5,0,0,0,0,0,0,0,0,0,0,0);
```

```
                                                    tcp_tick();
                                                }

                                            intransit =1;
                                            printf("continuing along wall\n");
                                            while(intransit==1){
                                               wdt_feed();
//                                             printf("desirednint=%lu, xint=%lu,
desiredeint=%lu,        yint=%lu\n", desirednint, xint, desiredeint, yint);
                                                                tcp_tick();
                                                                if (abs(desirednint-
xint)<3){
                                                                        if
(abs(desiredeint-yint)<3){

        intransit=0;
                                                                        }
                                                                }
                                                        }
                                                }
                                        }
                                }
                                else {
                                        printf("4");
                                        while(obstacle == 1){
                                                wdt_feed();
                                                printf("5");
                                                obstacle = 0;
                                                if (sensor[4]>1100){
                                                        printf("6");
                                                        if ((sensor[0]<1100)){
                                                                obstacle = 1;
                                                                for( chan = 0; chan < 5;
chan++ ) {
                                            sensor[chan] =  sensorinput(chan);
                                            }
                                            printf("Sensor Distances ");
                                                  printf("0: %u, 1: %u, 2: %u, 3: %u,
4: %u\n", sensor[0], sensor[1],
                                                                sensor[2], sensor[3],
sensor[4]);
                                            if (sensor[0]<600){
//                                             if ((sensor[1]<600)){
                                                                if (looksign == 1){
```

```
reltoabs(5,0,0,7-(sensor[0]/100),0,0);

flyto(16,1,2,genericint, genericdec, genericsign,

genericint2, genericdec2, genericsign2,

origdint, origddec, origdsign, 0,0,0);
//
flyto(16,4,2,5,0,0,7-(sensor[0]/100),0,0,0,0,0,0,0,0);
                                                        tcp_tick();
                                                    }
                                                    else{

reltoabs(5,0,0,7-(sensor[0]/100),0,1);

flyto(16,1,2,genericint, genericdec, genericsign,

genericint2, genericdec2, genericsign2,

origdint, origddec, origdsign, 0,0,0);
//
flyto(16,4,2,5,0,0,7-(sensor[0]/100),0,1,0,0,0,0,0,0);
                                                        tcp_tick();
                                                    }
//                                              }
                                                }
                                                else if (sensor[0]>600){
//                                                  if ((sensor[1]>600)){
                                                        if (looksign == 1){

reltoabs(5,0,0,(sensor[0]/100)-5,0,1);

flyto(16,1,2,genericint, genericdec, genericsign,

genericint2, genericdec2, genericsign2,

origdint, origddec, origdsign, 0,0,0);
//
flyto(16,4,2,5,0,0,(sensor[0]/100)-5,0,1,0,0,0,0,0,0);
                                                        tcp_tick();
                                                    }
                                                    else{

reltoabs(5,0,0,(sensor[0]/100)-5,0,0);
```

```
        flyto(16,1,2,genericint, genericdec, genericsign,
        genericint2, genericdec2, genericsign2,
        origdint, origddec, origdsign, 0,0,0);
//
        flyto(16,4,2,5,0,0,(sensor[0]/100)-5,0,0,0,0,0,0,0,0);
                                                                tcp_tick();
                                                            }
//                                                      }
                                                    }
                                                    else{
                                                        reltoabs(5,0,0,0,0,0);
                                                            flyto(16,1,3,genericint,
genericdec, genericsign,
                                                                genericint2,
genericdec2, genericsign2,
                                                                origdint,
origddec, origdsign, 0,0,0);
//
        flyto(16,4,3,5,0,0,0,0,0,0,0,0,0,0,0);
                                                        tcp_tick();
                                                    }
                                                intransit = 1;
                                                printf("continuing along wall\n");
                                                while(intransit==1){
                                                   wdt_feed();
//                                                 printf("desirednint=%lu, xint=%lu,
desiredeint=%lu,        yint=%lu\n", desirednint, xint, desiredeint, yint);
                                                                tcp_tick();
                                                                if (abs(desirednint-
xint)<3){
                                                                    if
(abs(desiredeint-yint)<3){

        intransit=0;
                                                                    }
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                reltoabs(5,0,0,0,0,0);
                                flyto(16,1,3,genericint, genericdec, genericsign,
```

```
                                        genericint2, genericdec2, genericsign2,
                                        origdint, origddec, origdsign, 0,0,0);
//                                  flyto(16,4,3,5,0,0,0,0,0,0,0,0,0,0,0);
                                    tcp_tick();
                        intransit = 1;
                        printf("extra\n");
                        while(intransit==1){
                           wdt_feed();
                                        tcp_tick();
//                                      printf("desirednint=%lu, xint=%lu, desiredeint=%lu,
        yint=%lu\n", desirednint, xint, desiredeint, yint);
                                        if (abs(desirednint-xint)<3){
                                                if (abs(desiredeint-yint)<3){
                                                        intransit=0;
                                                }
                                        }
                                    }
//                                  flyto(16,1,1,xint,xdec,xsign,yint,
//                                                  ydec,ysign,zint,zdec,zsign,
//                                                  orighdgint, orighdgdec,
orighdgsign);
//                                  intransit = 1;
//                                  printf("rotate\n");
//                                  while(intransit==1){
//                                          wdt_feed();
//                                          tcp_tick();
//                                          printf("orighdg=%u %lu.%08lu,
psi=%u %lu.%08lu\n",orighdgsign, orighdgint, orighdgdec,
//                                                  psisign, psiint, psidec);
//                                          if ((psiint-orighdgint)==0){
//                                                  if (((psidec-orighdgdec)/1000000)==0){
//                                                          intransit = 0;
//                                                  }
//                                          }
//                                  }
                                    flyto(16, 1, 5, orignint, orignsign, origeint,
origedec,
                                        origesign, origdint, origddec, origdsign,0,0,0);
                                    tcp_tick();
//                  }
//                  else if (sensor[1]<800){
//                          tcp_tick();
//                          orignint = desirednint;
//                          orignsign = desirednsign;
//                          origeint = desiredeint;
```

```
//                          origedec = desirededec;
//                          origesign = desiredesign;
//                          origdint = desireddint;
//                          origddec = desiredddec;
//                          origdsign = desireddsign;
//                          flyto(16,4,2,2,0,0,5,0,0,0,0,0,0,0,0);
//                          intransit =1;
//                          printf("obstacle front left\n");
//                        while(intransit==1){
//                          wdt_feed();
////                        printf("desirednint=%lu, xint=%lu, desiredeint=%lu,
        yint=%lu\n", desirednint, xint, desiredeint, yint);
//                                      tcp_tick();
//                                      if (abs(desirednint-xint)<3){
//                                              if (abs(desiredeint-yint)<3){
//                                                      intransit=0;
//                                              }
//                                      }
//                                  }
//                                  printf("front left done\n");
//                                  flyto(16, 1, 3, orignint, origndec, orignsign, origeint,
origedec,
//                                      origesign, origdint, origddec, origdsign, 0,0,0);
//                                  tcp_tick();
//                  }
//                  else if (sensor[3]<800){
//                          tcp_tick();
//                          orignint = desirednint;
//                          origndec = desiredndec;
//                          orignsign = desirednsign;
//                          origeint = desiredeint;
//                          origedec = desirededec;
//                          origesign = desiredesign;
//                          origdint = desireddint;
//                          origddec = desiredddec;
//                          origdsign = desireddsign;
//                          flyto(16,4,2,2,0,0,5,0,1,0,0,0,0,0,0);
//                          printf("obstacle front right\n");
//                          while(intransit==1){
//                                  wdt_feed();
////                        printf("desirednint=%lu, xint=%lu, desiredeint=%lu,
        yint=%lu\n", desirednint, xint, desiredeint, yint);
//                                      tcp_tick();
//                                      if (abs(desirednint-xint)<3){
//                                              if (abs(desiredeint-yint)<3){
//                                                      intransit=0;
```

```
//                                              }
//                                      }
//                              }
//                              flyto(16, 1, 3, orignint, origndec, orignsign, origeint,
origedec,
//                                      origesign, origdint, origddec, origdsign, 0,0,0);
//                              tcp_tick();
//              }
//              else if (sensor[2]<800){
//                      tcp_tick();
//                      orignint = desirednint;
//                      origndec = desiredndec;
//                      orignsign = desirednsign;
//                      origeint = desiredeint;
//                      origedec = desirededec;
//                      origesign = desiredesign;
//                      origdint = desireddint;
//                      origddec = desiredddec;
//                      origdsign = desireddsign;
//                      flyto(16,4,2,2,0,0,10,0,0,0,0,0,0,0,0);
//                      printf("obstacle only in front middle\n");
//                      while(intransit==1){
//                              wdt_feed();
////                    printf("desirednint=%lu, xint=%lu, desiredeint=%lu,
        yint=%lu\n", desirednint, xint, desiredeint, yint);
//                                      tcp_tick();
//                                      if (abs(desirednint-xint)<3){
//                                              if (abs(desiredeint-yint)<3){
//                                                      intransit=0;
//                                              }
//                                      }
//                              }
//                              flyto(16, 1, 3, orignint, origndec, orignsign, origeint,
origedec,
//                                      origesign, origdint, origddec, origdsign, 0,0,0);
//                              tcp_tick();
//              }

                }
                else if (sensor[1]<800){
                        tcp_tick();
                        orignint = desirednint;
                origndec = desiredndec;
                orignsign = desirednsign;
                origeint = desiredeint;
                origedec = desirededec;
```

```
                origesign = desiredesign;
                origdint = desireddint;
                origddec = desiredddec;
                origdsign = desireddsign;
                psiwantint = (((lookint*100000000)+lookdec)/15708)/10000;
                psiwantdec = ((((lookint*100000000)+lookdec)/15708)-
(psiwantint*10000))*10000;
                psiwantsign = looksign;
                decintadd(2,14450692,0,psiwantint,psiwantdec,psiwantsign);
                reltoabs(1,0,0,genericint,genericdec,genericsign);
                        flyto(16,1,2,genericint, genericdec, genericsign,
                                genericint2, genericdec2, genericsign2,
                                origdint, origddec, origdsign, 0,0,0);
//                      flyto(16,4,2,1,0,0,genericint,genericdec,genericsign,0,0,0,0,0,0);
                intransit = 1;
                while(intransit==1){
                wdt_feed();
//                              printf("desirednint=%lu, xint=%lu, desiredeint=%lu,
        yint=%lu\n", desirednint, xint, desiredeint, yint);
                                tcp_tick();
                                if (abs(desirednint-xint)<3){
                                        if (abs(desiredeint-yint)<3){
                                                intransit=0;
                                        }
                                }
                        }
                        flyto(16, 1, 5, orignint, origndec, orignsign, origeint, origedec,
                                origesign, origdint, origddec, origdsign,0,0,0);
                }
                else if (sensor[3]<800){
                        tcp_tick();
                        orignint = desirednint;
                origndec = desiredndec;
                orignsign = desirednsign;
                origeint = desiredeint;
                origedec = desirededec;
                origesign = desiredesign;
                origdint = desireddint;
                origddec = desiredddec;
                origdsign = desireddsign;
                        psiwantint = (((lookint*100000000)+lookdec)/15708)/10000;
                psiwantdec = ((((lookint*100000000)+lookdec)/15708)-
(psiwantint*10000))*10000;
                psiwantsign = looksign;
                decintadd(2,14450692,1,psiwantint,psiwantdec,psiwantsign);
                reltoabs(1,0,0,genericint,genericdec,genericsign);
```

```
                        flyto(16,1,2,genericint, genericdec, genericsign,
                                genericint2, genericdec2, genericsign2,
                                origdint, origddec, origdsign, 0,0,0);
//                      flyto(16,4,2,1,0,0,genericint,genericdec,genericsign,0,0,0,0,0,0);
                intransit = 1;
                while(intransit==1){
                wdt_feed();
//                              printf("desirednint=%lu, xint=%lu, desiredeint=%lu,
        yint=%lu\n", desirednint, xint, desiredeint, yint);
                                tcp_tick();
                                if (abs(desirednint-xint)<3){
                                        if (abs(desiredeint-yint)<3){
                                                intransit=0;
                                        }
                                }
                        }
                        flyto(16, 1, 5, orignint, orignsign, origeint, origedec,
                                origesign, origdint, origddec, origdsign, 0,0,0);
                }
                else if (sensor[0]<800){
                        tcp_tick();
                        orignint = desirednint;
                orignsign = desirednsign;
                origeint = desiredeint;
                origedec = desirededec;
                origesign = desiredesign;
                origdint = desireddint;
                origddec = desiredddec;
                origdsign = desireddsign;
                        psiwantint = (((lookint*100000000)+lookdec)/15708)/10000;
                psiwantdec = ((((lookint*100000000)+lookdec)/15708)-
(psiwantint*10000))*10000;
                psiwantsign = looksign;
                decintadd(0,83909963,0,psiwantint,psiwantdec,psiwantsign);
                reltoabs(1,0,0,genericint,genericdec,genericsign);
                        flyto(16,1,2,genericint, genericdec, genericsign,
                                genericint2, genericdec2, genericsign2,
                                origdint, origddec, origdsign, 0,0,0);
//                      flyto(16,4,2,1,0,0,genericint,genericdec,genericsign,0,0,0,0,0,0);
                intransit = 1;
                while(intransit==1){
                wdt_feed();
//                              printf("desirednint=%lu, xint=%lu, desiredeint=%lu,
        yint=%lu\n", desirednint, xint, desiredeint, yint);
                                tcp_tick();
```

```
                        if (abs(desirednint-xint)<3){
                                if (abs(desiredeint-yint)<3){
                                        intransit=0;
                                }
                        }
                }
                flyto(16, 1, 5, orignint, origndec, orignsign, origeint, origedec,
                        origesign, origdint, origddec, origdsign, 0,0,0);
        }
        else if (sensor[4]<800){
                tcp_tick();
                orignint = desirednint;
        origndec = desiredndec;
        orignsign = desirednsign;
        origeint = desiredeint;
        origedec = desirededec;
        origesign = desiredesign;
        origdint = desireddint;
        origddec = desiredddec;
        origdsign = desireddsign;
        psiwantint = (((lookint*100000000)+lookdec)/15708)/10000;
        psiwantdec = ((((lookint*100000000)+lookdec)/15708)-
(psiwantint*10000))*10000;
        psiwantsign = looksign;
        decintadd(0,83909963,1,psiwantint,psiwantdec,psiwantsign);
        reltoabs(1,0,0,genericint,genericdec,genericsign);
                flyto(16,1,2,genericint, genericdec, genericsign,
                        genericint2, genericdec2, genericsign2,
                        origdint, origddec, origdsign, 0,0,0);
//                      flyto(16,4,2,1,0,0,genericint,genericdec,genericsign,0,0,0,0,0,0);
        intransit = 1;
        while(intransit==1){
        wdt_feed();
//                              printf("desirednint=%lu, xint=%lu, desiredeint=%lu,
        yint=%lu\n", desirednint, xint, desiredeint, yint);
                              tcp_tick();
                              if (abs(desirednint-xint)<3){
                                      if (abs(desiredeint-yint)<3){
                                              intransit=0;
                                      }
                              }
                      }
                      flyto(16, 1, 5, orignint, origndec, orignsign, origeint, origedec,
                              origesign, origdint, origddec, origdsign, 0,0,0);
        }
  }
```

```
   if( udp_close( udp_socket ) == -1 ) {
        puts( "Error closing UDP socket!\n" ); }
   // Finished with the socket so return to pool
   if( udp_releasesocket( udp_socket ) == -1 ) {
        puts( "Error releasing UDP socket!\n" ); }

}

//called during tcp_tick if data is available to read from buffer
int
        udp_eventlistener( char handle, char event, int ip,
                short port, short buffindex, short datalen )
{
        char buf[1066];
        int i,n;
//      printf("UDP listener\n");
        if( handle != udp_socket ) {
                printf("DNS: not my handle!!!!");
                return( -1 ); }
//      printf("Packet from IP Address: %d.%d.%d.%d\n", IP1(ip),
//              IP2(ip), IP3(ip), IP4(ip) );
        UINT32 id, time, times, temp;
        UINT8 tempbuf[8];

        switch(event) {
                case UDP_EVENT_DATA:
                                RECEIVE_NETWORK_BUF(buf, datalen);
                                id = buf[11];

//                              printf("id = %lu ",id);

                                //get port from ground station to send data
                                if ((id == 1)&&(IP4(ip)!=5)){
                                        secip = ip;
                                        secport = port;

                                }
                                //stop sending open command when acknowledged by
AFCS
                                if ((id == 2)&&(IP4(ip)==5)){
                                        connected = 0;
                                }
                                //autopilot data to get desired position
                                if (id == 14){
                                        desired = 0;
                                        for (n=0; n<8; n++) {
```

```
                    tempbuf[7-n]=buf[n+28];
                }
                buftodec (tempbuf);
                desirednint = genericint;
                desiredndec = genericdec;
                desirednsign = genericsign;
                for (n=0; n<8; n++) {
                    tempbuf[7-n]=buf[n+36];
                }
                buftodec (tempbuf);
                desiredeint = genericint;
                desirededec = genericdec;
                desiredesign = genericsign;
                for (n=0; n<8; n++) {
                    tempbuf[7-n]=buf[n+44];
                }
                buftodec (tempbuf);
                desireddint = genericint;
                desiredddec = genericdec;
                desireddsign = genericsign;
                for (n=0; n<8; n++) {
                    tempbuf[7-n]=buf[n+92];
                }
                buftodec (tempbuf);
                desiredhdgint = genericint;
                desiredhdgdec = genericdec;
                desiredhdgsign = genericsign;
            }
//          //desired_pos data
//          if (id == 20){
//              desired = 0;
//              for (n=0; n<8; n++) {
//                  tempbuf[7-n]=buf[n+12];
//              }
//              buftodec (tempbuf);
//              desirednint = genericint;
//              desiredndec = genericdec;
//              desirednsign = genericsign;
//              for (n=0; n<8; n++) {
//                  tempbuf[7-n]=buf[n+20];
//              }
//              buftodec (tempbuf);
//              desiredeint = genericint;
//              desirededec = genericdec;
//              desiredesign = genericsign;
//              for (n=0; n<8; n++) {
```

```
//                                      tempbuf[7-n]=buf[n+28];
//                                  }
//                                  buftodec (tempbuf);
//                                  desireddint = genericint;
//                                  desiredddec = genericdec;
//                                  desireddsign = genericsign;
//                                  for (n=0; n<8; n++) {
//                                      tempbuf[7-n]=buf[n+36];
//                                  }
//                                  buftodec (tempbuf);
//                                  desiredhdgint = genericint;
//                                  desiredhdgdec = genericdec;
//                                  desiredhdgsign = genericsign;
//                              }

                                //gps data
//                              if (id == 12) {
////                                  printf("time = %u\n", buf[3]);
//                                  numsat = buf[13];
//                                  vdop = buf[15];
//                                  for (n=0; n<8; n++) {
//                                      tempbuf[7-n]=buf[n+20];
//                                  }
//                                  buftodec (tempbuf);
//                                  vel_nint = genericint;
//                                  vel_ndec = genericdec;
//                                  vel_nsign = genericsign;
//                                  printf("vel_n = ");
//                                  if (vel_nsign == 1){
//                                          printf("-");
//                                  }
//                                  printf("%lu.%08lu ",vel_nint, vel_ndec);
//                                  for (n=0; n<8; n++) {
//                                      tempbuf[7-n]=buf[n+28];
//                                  }
//                                  buftodec (tempbuf);
//                                  vel_eint = genericint;
//                                  vel_edec = genericdec;
//                                  vel_esign = genericsign;
//                                  printf("vel_e = ");
//                                  if (vel_esign == 1){
//                                          printf("-");
//                                  }
//                                  printf("%lu.%08lu ",vel_eint, vel_edec);
//                                  for (n=0; n<8; n++) {
//                                      tempbuf[7-n]=buf[n+36];
```

```
//                                      }
//                                      buftodec (tempbuf);
//                                      vel_dint = genericint;
//                                      vel_ddec = genericdec;
//                                      vel_dsign = genericsign;
//                                      printf("vel_d = ");
//                                      if (vel_dsign == 1){
//                                              printf("-");
//                                      }
//                                      printf("%lu.%08lu\n",vel_dint, vel_ddec);
//                              }
                                //state packet
                                if (id == 10) {
//                                      for (n=0; n<8; n++) {
//                                         tempbuf[7-n]=buf[n+12];
//                                      }
//                                      buftodec (tempbuf);
//                                      phiint = genericint;
//                                      phidec = genericdec;
//                                      phisign = genericsign;
//                                      printf("phi = ");
//                                      if (phisign == 1){
//                                              printf("-");
//                                      }
//                                      printf("%lu.%08lu ",phiint, phidec);
                                        for (n=0; n<8; n++) {
                                           tempbuf[7-n]=buf[n+20];
                                        }
                                        buftodec (tempbuf);
                                        thetaint = genericint;
                                        thetadec = genericdec;
                                        thetasign = genericsign;
//                                      printf("theta = ");
//                                      if (thetasign == 1){
//                                              printf("-");
//                                      }
//                                      printf("%lu.%08lu ",thetaint, thetadec);
                                        for (n=0; n<8; n++) {
                                           tempbuf[7-n]=buf[n+28];
                                        }
                                        buftodec (tempbuf);
                                        psiint = genericint;
                                        psidec = genericdec;
                                        psisign = genericsign;
//                                      printf("psi = ");
//                                      if (psisign == 1){
```

```
//                              printf("-");
//                          }
//                          printf("%lu.%08lu ",psiint, psidec);
                            for (n=0; n<8; n++) {
                               tempbuf[7-n]=buf[n+60];
                            }
                            buftodec (tempbuf);
                            xint = genericint;
                            xdec = genericdec;
                            xsign = genericsign;
//                          printf("x = ");
//                          if (xsign == 1){
//                              printf("-");
//                          }
//                          printf("%lu.%08lu ",xint, xdec);
                            for (n=0; n<8; n++) {
                               tempbuf[7-n]=buf[n+68];
                            }
                            buftodec (tempbuf);
                            yint = genericint;
                            ydec = genericdec;
                            ysign = genericsign;
//                          printf("y = ");
//                          if (ysign == 1){
//                              printf("-");
//                          }
//                          printf("%lu.%08lu ",yint, ydec);
                            for (n=0; n<8; n++) {
                               tempbuf[7-n]=buf[n+76];
                            }
                            buftodec (tempbuf);
                            zint = genericint;
                            zdec = genericdec;
                            zsign = genericsign;
//                          printf("z = ");
//                          if (zsign == 1){
//                              printf("-");
//                          }
//                          printf("%lu.%08lu\n",zint, zdec);
                            for (n=0; n<8; n++) {
                               tempbuf[7-n]=buf[n+84];
                            }
                            buftodec (tempbuf);
                            vxint = genericint;
                            vxdec = genericdec;
                            vxsign = genericsign;
```

```
//                                      printf("vx = ");
//                                      if (vxsign == 1){
//                                              printf("-");
//                                      }
//                                      printf("%lu.%08lu ",vxint, vxdec);
                                        for (n=0; n<8; n++) {
                                           tempbuf[7-n]=buf[n+92];
                                        }
                                        buftodec (tempbuf);
                                        vyint = genericint;
                                        vydec = genericdec;
                                        vysign = genericsign;
//                                      printf("vy = ");
//                                      if (vysign == 1){
//                                              printf("-");
//                                      }
//                                      printf("%lu.%08lu ",vyint, vydec);
                                        for (n=0; n<8; n++) {
                                           tempbuf[7-n]=buf[n+100];
                                        }
                                        buftodec (tempbuf);
                                        vzint = genericint;
                                        vzdec = genericdec;
                                        vzsign = genericsign;
//                                      printf("vz = ");
//                                      if (vzsign == 1){
//                                              printf("-");
//                                      }
//                                      printf("%lu.%08lu\n",vzint, vzdec);
                                        for (n=0; n<8; n++) {
                                           tempbuf[7-n]=buf[n+132];
                                        }
                                        buftodec (tempbuf);
                                        axint = genericint;
                                        for (n=0; n<8; n++) {
                                           tempbuf[7-n]=buf[n+140];
                                        }
                                        buftodec (tempbuf);
                                        ayint = genericint;
                                }
                                break;
                        default:
                                puts( "Unknown UDP event" );
        }

        return( 0 );
```

```
 }
int reltoabs(UINT32 oneint, UINT32 onedec, UINT32 onesign, UINT32 twoint,
                UINT32 twodec, UINT32 twosign)
{
        UINT32 angleint, xdiffint, ydiffint, distsqrint, xsqrint, ysqrint, xdiffsqr, den,
denint, anglesqrint, numint, angle;
        UINT32 angledec, xdiffdec, ydiffdec, distsqrdec, xsqrdec, ysqrdec, ydiff, dendec,
anglesqrdec, numdec;
        UINT8  anglesign, xdiffsign, ydiffsign, distsqrsign, xsqrsign, ysqrsign, densign;

        decintadd(oneint, onedec, 0, twoint, twodec, 0);
//      printf("generic %lu.%08lu\n",genericint, genericdec);
        angle = ((twoint*1000000+(twodec/100))/
                (genericint*1000+(genericdec/100000)))*(1571);
        if (onesign == 0){
                if (twosign == 0){
                        anglesign = 0;
                }
                else {
                        anglesign = 1;
                }
        }
        else {
                angle = 31415926-angle;
                if (twosign == 0){
                        anglesign = 0;
                }
                else {
                        anglesign = 1;
                }
        }
   angleint = angle/1000000;
   angledec = (angle-(angleint*1000000))*100;
        decintadd(angleint, angledec, anglesign, psiint, psidec, psisign);
        angleint = genericint;
        angledec = genericdec;
        anglesign = genericsign;
//      printf("angle =%u %lu.%08lu  loc = %u\n",anglesign, angleint, angledec, loc);
        if (angleint>2){
                if ((angledec>14159265)||(angleint>3)){
                        if (anglesign == 1){
                                anglesign = 0;
                        }
                        else{
                                anglesign = 1;
                        }
```

```
                        decintadd(6, 28318531, 0, angleint, angledec, 1);
                        angleint = genericint;
                        angledec = genericdec;
                }
        }
        if (angleint>0){
                if ((angledec>57079633)||(angleint>1)){
                        if (anglesign == 1){
                                xdiffsign = 1;
                                ydiffsign = 1;
                        }
                        else{
                                xdiffsign = 1;
                                ydiffsign = 0;
                        }
                        decintadd(3, 14159265, 0, angleint, angledec, 1);
                        angleint = genericint;
                        angledec = genericdec;
                }
                else{
                        if (anglesign == 1){
                                xdiffsign = 0;
                                ydiffsign = 1;
                        }
                        else{
                                xdiffsign = 0;
                                ydiffsign = 0;
                        }
                }
        }
        else{
                if (anglesign == 1){
                        xdiffsign = 0;
                        ydiffsign = 1;
                }
                else{
                        xdiffsign = 0;
                        ydiffsign = 0;
                }
        }

//      printf("angle =%u %lu.%08lu, xdiffsign=%u, ydiffsign=%u\n",anglesign,
angleint, angledec, xdiffsign, ydiffsign);

        decintmult(angleint, angledec, 0, 0, 63661977, 0);
        angleint = genericint;
```

```
        angledec = genericdec;

        decintmult(oneint, onedec, 0, oneint, onedec, 0);
        xsqrint = genericint;
        xsqrdec = genericdec;
        decintmult(twoint, twodec, 0, twoint, twodec, 0);
        ysqrint = genericint;
        ysqrdec = genericdec;
        decintadd(xsqrint, xsqrdec, 0, ysqrint, ysqrdec, 0);
        distsqrint = genericint;
        distsqrdec = genericdec;

        decintmult(angleint, angledec, 0, angleint, angledec, 0);
        anglesqrint=genericint;
        anglesqrdec=genericdec;
        decintadd(1, 0, 0, angleint, angledec, 1);
        decintmult(genericint, genericdec, 0, genericint, genericdec, 0);
        den =
(anglesqrint*100000000+anglesqrdec)/(genericint*10000+(genericdec/10000));
        denint = den/10000;
        dendec = (den-(denint*10000))*10000;
        xdiffsqr =
(distsqrint*1000000+(distsqrdec/100))/((denint+1)*100+(dendec/1000000));
        squareroot(xdiffsqr/10000, (xdiffsqr-((xdiffsqr/10000)*10000))*10000);
        xdiffint = genericint;
        xdiffdec = genericdec;
        decintmult(angleint, angledec, 0, xdiffint, xdiffdec, 0);
        numint = genericint;
        numdec = genericdec;
        decintadd(1, 0, 0, angleint, angledec, 1);
        denint = genericint;
        dendec = genericdec;
        if ((denint > 0)||((dendec/100000) > 0)){
                ydiff =
((numint*10000000+(numdec/10))/(denint*1000+(dendec/100000)));
                ydiffint = ydiff/10000;
                ydiffdec = (ydiff-(ydiffint*10000))*10000;
                decintadd(ydiffint, ydiffdec, ydiffsign, yint, ydec, ysign);
                genericint2 = genericint;
                genericdec2 = genericdec;
                genericsign2 = genericsign;
        }
        else{
                squareroot(distsqrint, distsqrdec);
                decintadd(genericint, genericdec, ydiffsign, yint, ydec, ysign);
                genericint2 = genericint;
```

```
                genericdec2 = genericdec;
                genericsign2 = genericsign;
        }
        decintadd(xdiffint, xdiffdec, xdiffsign, xint, xdec, xsign);
        return(0);
}

long buftodec (UINT8 guess[8])
{
        UINT16 e;
        UINT32 fint, fdec;
        UINT32 etempint, etempdec;
        UINT8 guessbin[64];
        UINT16 divider;
        int n,m;

        for (n=0; n<8; n++) {
           divider=1;
           //printf("%d ", guess[n]);
           for (m=0; m<8; m++) {
                divider *= 2;
                guessbin[(8*n)+m]=(guess[n])/(256/divider);
                //printf("%d",guessbin[(8*n)+m]);
                guess[n] -=(256/divider)*guessbin[(8*n)+m];
           }
           //printf(" ");
        }
        //printf("\n");
        divider = 1;
        e=0;
        for (n=1; n<12; n++){
                divider *= 2;
                e += guessbin[n]*(2048/divider);
        }
        //printf("e %u ", e);
        fdec=0;
        divider = 1;
        for (n=12; n<27; n++){
                divider *= 2;
                fdec += (100000000*guessbin[n])/divider;
        }
        //printf("fdec %ld ",fdec);
        fint = 1;
        if (e == 0){
                if (fdec == 0){
                        fint = 0;
```

```
                }
        }
        //printf("fint %ld ",fint);
        etempint = 1;
        etempdec = 0;
        if (e > 1023) {
                if ( e < 1053){
                        for (n=0; n<(e-1023); n++){
                                etempint *= 2;
                        }
                }
                else {
                        etempint = 1932735282; //max possible w/o error
                }
        }
        else if (e < 1023) {
                etempint = 0;
                etempdec = 100000000;
                for (n=0; n<(1023-e); n++){
                        etempdec /= 2;
                }
        }
        //printf("etempint %ld etempdec %ld ",etempint, etempdec);

        decintmult(etempint, etempdec, 0 , fint, fdec, 0);

        genericsign = 0;
        if (guessbin[0] == 1) {
                genericsign = 1;
        }
        //printf("decimal %ld.%08ld\n",genericint, genericdec);
        return(0);
}



int decintmult(UINT32 oneint, UINT32 onedec, UINT32 onesign, UINT32 twoint,
                UINT32 twodec, UINT32 twosign)
{
        UINT32 fronthalfone, backhalfone, fronthalftwo, backhalftwo;

        fronthalfone = onedec/10000;
        backhalfone = onedec-(fronthalfone*10000);
        fronthalftwo = twodec/10000;
        backhalftwo = twodec-(fronthalftwo*10000);
        genericdec = (fronthalfone*fronthalftwo)+((fronthalfone*backhalftwo)/10000)
```

```
                +((fronthalftwo*backhalfone)/10000);

        genericint = oneint*twoint;
        genericint += (oneint*fronthalftwo)/10000;
        genericint += (twoint*fronthalfone)/10000;
        genericdec += ((oneint*fronthalftwo)-(((oneint*fronthalftwo)/10000)*10000))
                *10000;
        genericdec += ((twoint*fronthalfone)-(((twoint*fronthalfone)/10000)*10000))
                *10000;
        genericdec += (oneint*backhalftwo);
        genericdec += (twoint*backhalfone);
        genericint += genericdec / 100000000;
        genericdec -= (genericdec / 100000000)*100000000;
        if ((onesign == 0)&&(twosign==0)){
                genericsign = 0;
        }
        else if ((onesign == 1)&&(twosign==1)){
                genericsign = 0;
        }
        else{
                genericsign = 1;
        }
        return(0);
}

int squareroot(unsigned long oneint, unsigned long onedec)
{
        UINT32 sqrtint, sqrtdec, tempint, tempdec;
        UINT32 adder;
        UINT8  under;
        short n;

        sqrtint = 0;
        sqrtdec = 0;
        adder = 1000;
        for (n=0; n < 4; n++){
                tempint = 0;
                while (tempint <= oneint){
                        sqrtint += adder;
                        decintmult(sqrtint, sqrtdec, 0, sqrtint, sqrtdec, 0);
                        tempint = genericint;
                        tempdec = genericdec;
                }
                sqrtint -= adder;
                adder /= 10;
        }
```

```
	adder = 10000000;
	for (n=0; n < 8; n++){
		tempdec = 0;
		tempint = 0;
		under = 1;
		while (under == 1){
			under = 0;
			if (tempint <= oneint){
				if ((tempdec <= onedec)||(tempint!=oneint)){
					sqrtdec += adder;
					decintmult(sqrtint, sqrtdec, 0, sqrtint, sqrtdec, 0);
					tempint = genericint;
					tempdec = genericdec;
					under = 1;
				}
			}
		}
		sqrtdec -= adder;
		adder /= 10;
	}
	genericint = sqrtint;
	genericdec = sqrtdec;


	return(0);
}

int decintadd(UINT32 oneint, UINT32 onedec, UINT32 onesign, UINT32 twoint,
		UINT32 twodec, UINT32 twosign)
{


	if (twosign==0){
		if (onesign==1){
			if (oneint>=twoint){
				oneint -= twoint;
			}
			else{
				oneint = twoint - oneint;
				onesign = 0;
			}
		}
		else{
			oneint += twoint;
		}
		if (onesign==1){
```

```
        if (onedec>=twodec){
                onedec -= twodec;
        }
        else if (oneint>0){
                        onedec += (100000000-twodec);
                        oneint -= 1;
        }
                else{
                        onedec = twodec-onedec;
                        onesign = 0;
        }
        }
        else{
                if (onedec<(100000000-twodec)){
                onedec += twodec;
        }
        else if (onedec==(100000000-twodec)){
                        onedec = 0;
                        oneint += 1;
        }
                else{
                        onedec -= (100000000-twodec);
                        oneint += 1;
        }
        }
}
else{
        if (onesign==0){
                if (oneint>=twoint){
                        oneint -= twoint;
                }
                else{
                        oneint = twoint - oneint;
                        onesign = 1;
                }
        }
        else{
                oneint += twoint;
        }
        if (onesign==0){
        if (onedec>=twodec){
                onedec -= twodec;
        }
        else if (oneint>0){
                        onedec += (100000000-twodec);
                        oneint -= 1;
```

```
                }
                        else{
                                onedec = twodec-onedec;
                                onesign = 1;
                }
                }
                else{
                        if (onedec<(100000000-twodec)){
                        onedec += twodec;
                }
                else if (onedec==(100000000-twodec)){
                                onedec = 0;
                                oneint += 1;
                }
                        else{
                                onedec -= (100000000-twodec);
                                oneint += 1;
                }
                }
        }
        genericsign = onesign;
        genericint = oneint;
        genericdec = onedec;
        return(0);
}
int dectobuf(UINT32 oneint, UINT32 onedec, UINT32 onesign)
{
        UINT16 e;
        UINT32 dectemp, etemp;
        UINT32 inttemp, ftemp, fdectemp, finttemp;
        UINT8 guessbin[64];
        UINT16 divider;

        int n,m;


        e = 1023;
        inttemp = 1;
        dectemp = 0;
        if (oneint > 1){
                while (oneint >= inttemp){
                        e += 1;
                        inttemp *= 2;
                }
                inttemp /= 2;
                e -= 1;
```

```
	}
	else if (oneint == 1){
		e = 1023;
	}
	else if ((oneint == 0)&&(onedec!=0)){
		inttemp = 0;
		dectemp = 100000000;
		while (onedec < dectemp){
			e -= 1;
			dectemp /= 2;
		}
	}
	else{
		e = 0;
	}
	divider = 1;
	guessbin[0] = onesign;
	etemp = e;
	for (n=1; n<12; n++){
		divider *= 2;
	guessbin[n]=e /(2048/divider);
	e -=(2048/divider)*guessbin[n];
	}
	fdectemp = 0;
	finttemp = 1;
	divider = 1;
	for (n=12; n<27; n++){
		divider *= 2;
		fdectemp += 100000000/divider;
		decintmult(inttemp, dectemp, 0, finttemp, fdectemp, 0);
//		printf("%lu.%04lu*%lu.%04lu = %lu.%08lu\n",inttemp, dectemp,
finttemp,
//			fdectemp, genericint, genericdec);
//		printf("dec %lu\n",100000000/divider);
		if (genericint>oneint){
			guessbin[n] = 0;
			fdectemp -= 100000000/divider;
		}
		else if ((genericint==oneint)&&(genericdec>onedec)){
			guessbin[n] = 0;
			fdectemp -= 100000000/divider;
		}
		else{
			guessbin[n] = 1;
//			printf("used");
		}
```

```
		}

		for (n=27; n<64; n++){
			guessbin[n] = 0;
		}
		for (n=0; n<64; n++){
			//printf("%d",guessbin[n]);
			if (((n+1)%8)==0){
				//printf(" ");
			}
		}
		//printf("\n");
		for (n=0; n<8; n++){
			divider = 1;
			bytbuf[7-n]=0;
			for (m=0; m<8; m++){
				divider *= 2;
				bytbuf[7-n]+=guessbin[(n*8)+m]*(256/divider);
			}
			//printf("bytbuf %u n %d\n", bytbuf[n], n);
		}
//		printf("%lu.%04lu = %u*1.%1u%1u%1u%1u%1u%1u\n", oneint, onedec, etemp,
guessbin[12],
//			guessbin[13], guessbin[14], guessbin[15], guessbin[16], guessbin[17]);
		return(0);
}
int flyto(unsigned char failsafe, unsigned char posmode, unsigned char hdgmode,
		unsigned long nint, unsigned long ndec, unsigned char nsign,
		unsigned long eint, unsigned long edec, unsigned char esign,
		unsigned long dint, unsigned long ddec, unsigned char dsign,
		unsigned long hdgint, unsigned long hdgdec, unsigned char hdgsign)
{
		UINT8 *buf_ptr;
   UINT8 *buf;
   char traveltime1, traveltime2;
   UINT32 desirednintprev, desiredeintprev, desiredndecprev, desirededecprev;

//   random += 1;
//   if (random == 200){
//		random = 0;
//   }

   printf("flyto\n");

   //define flyto packet and send
   buf = buf_ptr = OTCP_TXBUF + UDP_APP_OFFSET;
```

```
    int n;
    for (n=0;n<7;n++){  //unimportant time padding
          *buf_ptr++ = 0;
    }
    *buf_ptr++ = 0; //random time change to prevent rejection of duplicate packets
    *buf_ptr++ = 0; //flyto command
    *buf_ptr++ = 0;
    *buf_ptr++ = 0;
    *buf_ptr++ = failsafe; //Identifies as packet Id "flyto" or "failsafe"
    *buf_ptr++ = 0;  //pos_mode
    *buf_ptr++ = posmode;
    *buf_ptr++ = 0;  //hdg_mode
    *buf_ptr++ = hdgmode;
    traveltime1 = -500>>8;
    traveltime2 = -500-(traveltime1<<8);
    *buf_ptr++ = traveltime1;  //transit_time in binary for -500 which means 5 m/s travel
speed
    *buf_ptr++ = traveltime2;
    *buf_ptr++ = 0; // padding/timeout time for failsafe
          *buf_ptr++ = 1;
    dectobuf(nint, ndec, nsign);
    for (n=0;n<8;n++){  //x
          *buf_ptr++ = bytbuf[n];
    }
    dectobuf(eint, edec, esign);
    for (n=0;n<8;n++){  //y
          *buf_ptr++ = bytbuf[n];
    }
    dectobuf(dint, ddec, dsign);
    for (n=0;n<8;n++){  //z
                  *buf_ptr++ = bytbuf[n];
          }
          dectobuf(hdgint, hdgdec, hdgsign);
    for (n=0;n<8;n++){  //hdg
          *buf_ptr++ = bytbuf[n];
    }
    desired = 1;
    while (desired == 1){
          wdt_feed();
          tcp_tick();
    }
    desirednintprev = desirednint;
    desiredeintprev = desiredeint;
    desiredndecprev = desiredndec;
    desirededecprev = desirededec;
```

```
        udp_send( udp_socket, remote_ip, 2002, buf, NETWORK_TX_BUFFER_SIZE -
UDP_APP_OFFSET, 52 );
   udp_send( udp_socket, secip, secport, buf, NETWORK_TX_BUFFER_SIZE -
UDP_APP_OFFSET, 52 );
        desired = 1;
   while (desired == 1){
        wdt_feed();
        tcp_tick();
        desired = 1;
                if (desirednintprev != desirednint){
                        desired = 0;
                }
                if (desiredeintprev != desiredeint){
                        desired = 0;
                }
                if (desiredndecprev != desiredndec){
                        desired = 0;
                }
                if (desirededecprev != desirededec){
                        desired = 0;
                }
   }

        return(0);
}
int lookat(UINT8 lookmode, UINT32 pitchint, UINT32 pitchdec, UINT8 pitchsign,
        UINT32 yawint, UINT32 yawdec, UINT8 yawsign)
{
        UINT8 *buf_ptr;
   UINT8 *buf;

//    random += 1;
//    if (random == 200){
//        random = 0;
//    }


        //define platform packet and send
   buf_ptr = 0;
   buf = buf_ptr = OTCP_TXBUF + UDP_APP_OFFSET;
   int n;
   for (n=0;n<7;n++){  //unimportant time padding
        *buf_ptr++ = 0;
   }
   *buf_ptr++ = 0; //random time change to prevent rejection of duplicate packets
   *buf_ptr++ = 0; //platform command
```

```
    *buf_ptr++ = 0;
    *buf_ptr++ = 0;
    *buf_ptr++ = 19;
    *buf_ptr++ = 0;  //mode
    *buf_ptr++ = lookmode;
    for (n=14;n<28;n++){  //padding, relative_roll
        *buf_ptr++ = 0;
    }
    dectobuf(pitchint, pitchdec, pitchsign);
        for (n=0;n<8;n++){  //relative_pitch
        *buf_ptr++ = bytbuf[n];
    }
    dectobuf(yawint, yawdec, yawsign);
        for (n=0;n<8;n++){  //relative_yaw
            *buf_ptr++ = bytbuf[n];
        }
        for (n=0;n<8;n++){  //absolute_roll
        *buf_ptr++ = 0;
    }
    dectobuf(pitchint, pitchdec, pitchsign);
        for (n=0;n<8;n++){  //absolute_pitch
        *buf_ptr++ = bytbuf[n];
    }
    dectobuf(yawint, yawdec, yawsign);
        for (n=0;n<8;n++){  //absolute_yaw
            *buf_ptr++ = bytbuf[n];
        }
        for (n=68;n<92;n++){  //n, e, d
            *buf_ptr++ = 0;
        }
        udp_send( udp_socket, remote_ip, 2002, buf, NETWORK_TX_BUFFER_SIZE
            - UDP_APP_OFFSET, 92 );
        tcp_tick();
        udp_send( udp_socket, secip, secport, buf, NETWORK_TX_BUFFER_SIZE
            - UDP_APP_OFFSET, 92 );
        tcp_tick();
        return(0);
}
unsigned short sensorinput(unsigned char loc)
{
        unsigned short dist;
        UINT32 xsqrint, xsqrdec, ysqrint, ysqrdec, distsqrint, distsqrdec;
        UINT32 angleint, angledec, numint, numdec, denint, dendec;
        UINT8  anglesign, flipsign;
        UINT32 xfrontobsint, xbackobsint, yleftobsint, yrightobsint, xdiffint, ydiffint;
```

```
	UINT32 xfrontobsdec, xbackobsdec, yleftobsdec, yrightobsdec, xdiffdec,
ydiffdec;
	UINT8  xfrontobssign, xbackobssign, yleftobssign, yrightobssign, xdiffsign,
ydiffsign;

	UINT32 xtemp, xtempint, xtempdec, ytemp, ytempint, ytempdec;
	UINT8  xtempsign, ytempsign, xdirsign, ydirsign, calculate, checkx, checky;
	int n;

	xfrontobsint = 50;
	xfrontobsdec = 0;
	xfrontobssign = 0;
	xbackobsint = 100;
	xbackobsdec = 0;
	xbackobssign = 0;
	yleftobsint = 50;
	yleftobsdec = 0;
	yleftobssign = 0;
	yrightobsint = 100;
	yrightobsdec = 0;
	yrightobssign = 0;

	decintadd(psiint, psidec, psisign, lookint, lookdec, looksign);
	angleint = genericint;
	angledec = genericdec;
	anglesign = genericsign;
//	printf("before %u %lu.%08lu\n",anglesign, angleint, angledec);
	if (loc == 0){
		decintadd(angleint, angledec, anglesign, 0, 87266463, 1);
		angleint = genericint;
		angledec = genericdec;
		anglesign = genericsign;
	}
	if (loc == 1){
		decintadd(angleint, angledec, anglesign, 0, 43633231, 1);
		angleint = genericint;
		angledec = genericdec;
		anglesign = genericsign;
	}
	if (loc == 3){
		decintadd(angleint, angledec, anglesign, 0, 43633231, 0);
		angleint = genericint;
		angledec = genericdec;
		anglesign = genericsign;
	}
	if (loc == 4){
```

```
            decintadd(angleint, angledec, anglesign, 0, 87266463, 0);
            angleint = genericint;
            angledec = genericdec;
            anglesign = genericsign;
        }


//      printf("angle =%u %lu.%08lu  loc = %u\n",anglesign, angleint, angledec, loc);

        if (angleint>2){
            if ((angledec>14159265)||(angleint>3)){
                if (anglesign == 1){
                    anglesign = 0;
                }
                else{
                    anglesign = 1;
                }
                decintadd(6, 28318531, 0, angleint, angledec, 1);
                angleint = genericint;
                angledec = genericdec;
            }
        }
        if (angleint>0){
            if ((angledec>57079633)||(angleint>1)){
                if (anglesign == 1){
                    xdiffsign = 1;
                    ydiffsign = 1;
                }
                else{
                    xdiffsign = 1;
                    ydiffsign = 0;
                }
                decintadd(3, 14159265, 0, angleint, angledec, 1);
                angleint = genericint;
                angledec = genericdec;
            }
            else{
                if (anglesign == 1){
                    xdiffsign = 0;
                    ydiffsign = 1;
                }
                else{
                    xdiffsign = 0;
                    ydiffsign = 0;
                }
            }
```

```
		}
		else{
			if (anglesign == 1){
				xdiffsign = 0;
				ydiffsign = 1;
			}
			else{
				xdiffsign = 0;
				ydiffsign = 0;
			}
		}

//		printf("loc = %u, angle =%u %lu.%08lu, xdiffsign=%u,
ydiffsign=%u\n",loc,anglesign, angleint, angledec, xdiffsign, ydiffsign);

		decintmult(angleint, angledec, 0, 0, 63661977, 0);
		angleint = genericint;
		angledec = genericdec;
		if (xsign == 0){
			flipsign = 1;
		}
		else{
			flipsign = 0;
		}
		if (xdiffsign == 0){
			decintadd(xfrontobsint, xfrontobsdec, xfrontobssign, xint, xdec, flipsign);
		}
		else{
			decintadd(xbackobsint, xbackobsdec, xbackobssign, xint, xdec, flipsign);
		}
		xdiffint = genericint;
		xdiffdec = genericdec;
		xdirsign = genericsign;
		if (ysign == 0){
			flipsign = 1;
		}
		else{
			flipsign = 0;
		}
		if (ydiffsign == 0){
			decintadd(yleftobsint, yleftobsdec, yleftobssign, yint, ydec, flipsign);
		}
		else{
			decintadd(yrightobsint, yrightobsdec, yrightobssign, yint, ydec, flipsign);
		}
		ydiffint = genericint;
```

```
	ydiffdec = genericdec;
	ydirsign = genericsign;
//	printf("xdiffint = %u  %lu.%08lu, xdirsign = %u\n", xdiffsign, xdiffint, xdiffdec,
xdirsign);
//	printf("ydiffint = %u %lu.%08lu, ydirsign = %u\n", ydiffsign, ydiffint, ydiffdec,
ydirsign);
	dist = 1219;
	checkx=1;
	checky=1;
	if (ysign == 1){
		if (yleftobssign == 1){
			if (yint<(yleftobsint+2)){
				if ((ydec<yleftobsdec)||(yint<(yleftobsint+1))){
					if (yrightobssign == 1){
						if (yint>(yrightobsint-2)){
							if
((ydec>yrightobsdec)||(yint>(yrightobsint-1))){
								checky = 0;
							}
						}
					}
					else{
						checky = 0;
					}
				}
			}
		}
	}
	else {
		if (yrightobssign == 0){
			if (yint<(yrightobsint+2)){
				if ((ydec<yrightobsdec)||(yint<(yrightobsint+1))){
					if (yleftobssign == 0){
						if (yint>(yleftobsint-2)){
							if
((ydec>yleftobsdec)||(yint>(yleftobsint-1))){
								checky = 0;
							}
						}
					}
					else{
						checky = 0;
					}
				}
			}
		}
```

```
	}
	if (xsign == 1){
		if (xfrontobssign == 1){
			if (xint<(xfrontobsint+2)){
				if ((xdec<xfrontobsdec)||(xint<(xfrontobsint+1))){
					if (xbackobssign == 1){
						if (xint>(xbackobsint-2)){
							if
((xdec>xbackobsdec)||(xint>(xbackobsint-1))){
								checkx = 0;
							}
						}
					}
					else{
						checkx = 0;
					}
				}
			}
		}
	}
	else {
		if (xbackobssign == 0){
			if (xint<(xbackobsint+1)){
				if (xdec<xbackobsdec){
					if (xfrontobssign == 0){
						if (xint>(xfrontobsint-2)){
							if (xdec>xfrontobsdec){
								checkx = 0;
							}
							if (xint>(xfrontobsint-1)){
								checkx = 0;
							}
						}
					}
					else{
						checkx = 0;
					}
				}
				if (xint<(xbackobsint+0)){
					if (xfrontobssign == 0){
						if (xint>(xfrontobsint-2)){
							if (xdec>xfrontobsdec){
								checkx = 0;
							}
							if (xint>(xfrontobsint-1)){
								checkx = 0;
```

```
                                                }
                                        }
                                }
                                else{
                                        checkx = 0;
                                }
                        }
                }
        }
        if ((checkx==0)&&(checky==0)){
                for(n=0; n<1; n++){
                        printf("crash!!!\n");
                }
        }
        if (xdirsign != xdiffsign){
                checkx = 0;
        }
        if (ydirsign != ydiffsign){
                checky = 0;
        }
        if (xdiffint<14){
                if (checkx==1){
                        decintmult(angleint, angledec, 0, xdiffint, xdiffdec, 0);
                        numint = genericint;
                        numdec = genericdec;
                        decintadd(1, 0, 0, angleint, angledec, 1);
                        denint = genericint;
                        dendec = genericdec;
                        if ((denint < 0)||(dendec > 99999)){
                                ytemp =
((numint*10000000+(numdec/10))/(denint*1000+(dendec/100000)));
                        }
                        else{
                                ytemp = 1000000;
                        }
                        ytempint = ytemp/10000;
                        ytempdec = ytemp-(ytempint*10000);
                        decintadd(ytempint, ytempdec, ydiffsign, yint, ydec, ysign);
                        ytempint = genericint;
                        ytempdec = genericdec;
                        ytempsign = genericsign;
        //                      printf("ytemp = %u %lu.%08lu\n",ytempsign, ytempint,
ytempdec);
                        calculate = 0;
                        if (ytempsign == 1){
```

```
                        if (yleftobssign == 1){
                                if (ytempint<(yleftobsint+1)){
                                        if
((ytempdec<yleftobsdec)||(ytempint<yleftobsint)){
                                                if (yrightobssign == 1){
                                                        if (ytempint>(yrightobsint-
1)){
                                                                if
((ytempdec>yrightobsdec)||(ytempint>yrightobsint)){
                                                                        calculate = 1;
                                                                }
                                                        }
                                                }
                                                else{
                                                        calculate = 1;
                                                }
                                        }
                                }
                        }
                }
                else {
                        if (yrightobssign == 0){
                                if (ytempint<(yrightobsint+1)){
                                        if
((ytempdec<yrightobsdec)||(ytempint<yrightobsint)){
                                                if (yleftobssign == 0){
                                                        if (ytempint>(yleftobsint-1)){
                                                                if
((ytempdec>yleftobsdec)||(ytempint>yleftobsint)){
                                                                        calculate = 1;
                                                                }
                                                        }
                                                }
                                                else{
                                                        calculate = 1;
                                                }
                                        }
                                }
                        }
                }
                if (calculate == 1){
                        decintmult(xdiffint, xdiffdec, 0, xdiffint, xdiffdec, 0);
                        xsqrint = genericint;
                        xsqrdec = genericdec;
                        if (ysign == 0){
```

```
                                        decintadd(ytempint, ytempdec, ytempsign, yint,
ydec, 1);
                                }
                                else{
                                        decintadd(ytempint, ytempdec, ytempsign, yint,
ydec, 0);
                                }
                                decintmult(genericint, genericdec, 0, genericint, genericdec,
0);
                                ysqrint = genericint;
                                ysqrdec = genericdec;
                                decintadd(xsqrint, xsqrdec, 0, ysqrint, ysqrdec, 0);
                                distsqrint = genericint;
                                distsqrdec = genericdec;
                                squareroot(distsqrint, distsqrdec);
        //                              printf("sqrt(%lu.%08lu) = %lu.%08lu\n", distsqrint,
distsqrdec, genericint, genericdec);
                                if (((genericint*100)+(genericdec/1000000))<dist){
                                        dist = (genericint*100)+(genericdec/1000000);
                                }
                        }
                }
        }

        if (ydiffint<14){
                if (checky==1){
                        decintadd(1, 0, 0, angleint, angledec, 1);
                        decintmult(genericint, genericdec, 0, ydiffint, ydiffdec, 0);
                        numint = genericint;
                        numdec = genericdec;
                        if (angledec > 99999){
                                xtemp =
((numint*10000000+(numdec/10))/((angleint*1000)+(angledec/100000)));
                        }
                        else if (angleint < 0){
                                xtemp =
((numint*10000000+(numdec/10))/((angleint*1000)));
                        }
                        else{
                                xtemp = 1000000;
                        }
                        xtempint = xtemp/10000;
                        xtempdec = xtemp-(xtempint*10000);
                        decintadd(xtempint, xtempdec, xdiffsign, xint, xdec, xsign);
                        xtempint = genericint;
                        xtempdec = genericdec;
```

```
                    xtempsign = genericsign;
                    calculate = 0;
                    if (xtempsign == 1){
                        if (xfrontobssign == 1){
                            if (xtempint<(xfrontobsint+1)){
                                if
((xtempdec<xfrontobsdec)||(xtempint<xfrontobsint)){
                                    if (xbackobssign == 1){
                                        if
(xtempint>(xbackobsint-1)){
                                            if
((xtempdec>xbackobsdec)||(xtempint>xbackobsint)){
                                                calculate = 1;
                                            }
                                        }
                                    }
                                    else{
                                        calculate = 1;
                                    }
                                }
                            }
                        }
                    }
                    else {
                        if (xbackobssign == 0){
                            if (xtempint<(xbackobsint+1)){
                                if
((xtempdec<xbackobsdec)||(xtempint<xbackobsint)){
                                    if (xfrontobssign == 0){
                                        if (xtempint>(xfrontobsint-
1)){
                                            if
((xtempdec>xfrontobsdec)||(xtempint>xfrontobsint)){
                                                calculate = 1;
                                            }
                                        }
                                    }
                                    else{
                                        calculate = 1;
                                    }
                                }
                            }
                        }
                    }
                    if (calculate == 1){
                        if (xsign == 0){
```

```
                                        decintadd(xtempint, xtempdec, xtempsign, xint,
xdec, 1);
                                }
                                else{
                                        decintadd(xtempint, xtempdec, xtempsign, xint,
xdec, 0);
                                }
                               decintmult(genericint, genericdec, 0, genericint, genericdec,
0);
                                xsqrint = genericint;
                                xsqrdec = genericdec;
                                decintmult(ydiffint, ydiffdec, 0, ydiffint, ydiffdec, 0);
                                ysqrint = genericint;
                                ysqrdec = genericdec;
                                decintadd(xsqrint, xsqrdec, 0, ysqrint, ysqrdec, 0);
                                distsqrint = genericint;
                                distsqrdec = genericdec;
                                squareroot(distsqrint, distsqrdec);
        //                              printf("sqrt(%lu.%08lu) = %lu.%08lu\n", distsqrint,
distsqrdec, genericint, genericdec);
                                if (((genericint*100)+(genericdec/1000000))<dist){
                                        dist = (genericint*100)+(genericdec/1000000);
                                }
                        }
                }
        }
//      printf("distance %u\n",dist);
        return(dist);

}
```

APPENDIX B

REAL WORLD OBSTACLE AVOIDANCE CODE

```
#include <wf5282.h>        // GPTPort, ...
#include <stdio.h>         // printf
#include <stdlib.h>                    // strtod
#include <opentcp.h>       // NIF and all other OpenTcp


//------------- Implementation ------------------------------------------
int udp_eventlistener(char , char , int , short , short , short );
char udp_socket;
UINT32 *remote_ip;
char connected, desired, intransit;
UINT8  genericsign, genericsign2, random;
UINT32 genericint, genericint2;
UINT32 genericdec, genericdec2;

//UINT8  vel_nsign, vel_esign, vel_dsign;
//UINT8  numsat, vdop;
//UINT32 vel_nint, vel_eint, vel_dint;
//UINT32 vel_ndec, vel_edec, vel_ddec;
UINT8  vxsign, vysign, vzsign, thetasign, psisign;
UINT8  xsign, ysign, zsign;
UINT32 vxint, vyint, vzint, thetaint, psiint, axint, ayint;
UINT32 xint, yint, zint;
UINT32 vxdec, vydec, vzdec, thetadec, psidec;
UINT32 xdec, ydec, zdec;
UINT8 bytbuf[8];

UINT8  desirednsign, desiredesign, desireddsign, desiredhdgsign, looksign;
UINT32 desirednint, desiredeint, desireddint, desiredhdgint, lookint;
UINT32 desiredndec, desirededec, desiredddec, desiredhdgdec, lookdec;
UINT8  tempnsign, tempesign, temphdgsign;
UINT32 tempnint, tempeint, temphdgint;
UINT32 tempndec, tempedec, temphdgdec;

UINT32 *secip;
short secport;

                    // TYPE: Main
int                  // return error code (not used)
   main( void )          // main function called from crt0.s -> from dBUG
{
   UINT8  wallside, inloop;
   UINT32 look;
         UINT8 *buf_ptr;
   UINT8 *buf;
   UINT16 count;
```

```
   UINT8 chan;
   unsigned short sensor[5], distance, diff, diffprev;
   UINT8  orignsign, origesign, origdsign, orighdgsign, tempnsign, tempesign,
deltalooksign;
        UINT32 orignint, origeint, origdint, orighdgint, tempnint, tempeint, deltalookint;
        UINT32 origndec, origedec, origddec, orighdgdec, tempndec, tempedec,
deltalookdec;

        UINT8 lookmode, obstacle;
        UINT32 pitchint;
        UINT32 pitchdec;
        UINT8  pitchsign;
        UINT32 psiwantint;
        UINT32 psiwantdec;
        UINT8  psiwantsign;
        UINT32 n;
        UINT32 xsqrint, ysqrint, deltanint, deltaeint, prop, propint, newnint, neweint;
        UINT32 xsqrdec, ysqrdec, deltandec, deltaedec, propdec, newndec, newedec;
        UINT8  xsqrsign, ysqrsign, deltansign, deltaesign, propsign, newnsign, newesign;
        random = 0;
        UINT32 xsin50, xcos50, xsin25, xcos25;
        UINT32 deltax, deltay;

        wdt_timeout_set( 0x0FA0 );
   //Initialize TCP/IP subsystem
   tcpip_init(0);
   //Open a UDP socket for communication
   udp_socket = udp_getsocket(0 , udp_eventlistener, 0);
   if( udp_socket == -1 ) {
        printf("DNS: No free UDP sockets!! \r\n");
        return; }
   //Bind socket to port and open for communication
   if( udp_open( udp_socket, 2002 ) >= 0 ) {
        puts( "Opened UDP socket on local port 2002\n" ); }
   //Send Open command to AFCS in loop until acknowledge packet received
   remote_ip = make_ip( 10, 255, 0, 5 );
        connected = 1;
        while(connected==1){
                wdt_feed();
           buf = buf_ptr = OTCP_TXBUF + UDP_APP_OFFSET;
           *buf_ptr++ = 1;  //unimportant time padding
           *buf_ptr++ = 2;
           *buf_ptr++ = 3;
           *buf_ptr++ = 4;
           *buf_ptr++ = 9;
           *buf_ptr++ = 6;
```

```
            *buf_ptr++ = 7;
            *buf_ptr++ = 8;
            *buf_ptr++ = 0;  //Identifies as packet Id "Open"
            *buf_ptr++ = 0;
            *buf_ptr++ = 0;
            *buf_ptr++ = 1;
            *buf_ptr++ = 0;  //extra byte needed for this controller
            udp_send( udp_socket, remote_ip, 2002, buf,
NETWORK_TX_BUFFER_SIZE - UDP_APP_OFFSET, 13 );
            tcp_tick();
         }
         ad_init(                // Initialize the AD converter
               AD_SAMPLE_16TICS,      // 16 Ticks (8 microseconds sampling time)
               AD_8CHAN_8SCAN );     // Get average of 8 most recent scans

         count=0;

   while(1){
         wdt_feed();
//       count += 1;
//       printf("count = %lu ",count);
         tcp_tick();

//          convertxytone(1,0,0,0,0,0);
//          printf("n = %u %lu.%08lu, e = %u %lu.%08lu\n",genericsign, genericint,
genericdec,
//                genericsign2, genericint2, genericdec2);
//       vxsign = 0;
//       vxint = 1;
//       vxdec = 0;
//       vysign = 0;
//       vyint = 1;
//       vydec = 0;
//       vzsign = 0;
//       vzint = 0;
//       vzdec = 0;

//       printf("vx %d %2lu.%08lu, vy %d %2lu.%08lu, vz %d %2lu.%08lu\n",
//                vxsign, vxint, vxdec,
//                vysign, vyint, vydec, vzsign, vzint, vzdec);
         //when faster than 1 m/s point platform in that direction
         look = 0;
         lookint = 0;
         lookdec = 0;
         looksign = 0;
         if (((vxint+vyint)>0)||((vxdec+vydec)>100000000)){
```

```
                decintadd(vxint, vxdec, 0, vyint, vydec, 0);
//              printf("generic %lu.%08lu\n",genericint, genericdec);
                look = ((vyint*1000000+(vydec/100))/
                        (genericint*1000+(genericdec/100000)))*(1571);
                if (vxsign == 0){
                        if (vysign == 0){
                                looksign = 0;
                        }
                        else {
                                looksign = 1;
                        }
                }
                else {
                                look = 31415926-look;
                        if (vysign == 0){
                                looksign = 0;
                        }
                        else {
                                looksign = 1;
                        }
                }
                   lookint = look/1000000;
                   lookdec = (look-((look/1000000)*1000000))*100;
                        if (thetasign == 0){
                                lookat(0, thetaint, thetadec, 1, lookint, lookdec, looksign);
                        }
                        else{
                                lookat(0, thetaint, thetadec, 0, lookint, lookdec, looksign);
                        }
//                 lookat(0, 0, 0, 0, 0, 0, 0);
                        if ((lookint>0)&&(lookdec>57079633)){
                                lookint = 1;
                                lookdec = 57079633;
                        }
        }
      else if (thetasign == 0){
                        lookat(0, thetaint, thetadec, 1, 0,0,0);
                        lookint = 0;
                        lookdec = 0;
                        looksign = 0;
                }
                else{
                        lookat(0, thetaint, thetadec, 0, 0,0,0);
                        lookint = 0;
                        lookdec = 0;
                        looksign = 0;
```

```
                }
        //read sensors in cm
        for( chan = 0; chan < 5; chan++ ) {
           sensor[chan] =  (ad_get( chan )*1189/1023)+30;
        }
        buf = buf_ptr = OTCP_TXBUF + UDP_APP_OFFSET;
              *buf_ptr++ = 1;  //unimportant time padding
              *buf_ptr++ = 2;
              *buf_ptr++ = 3;
              *buf_ptr++ = 4;
              *buf_ptr++ = 9;
              *buf_ptr++ = 6;
              *buf_ptr++ = 7;
              *buf_ptr++ = 8;
              *buf_ptr++ = 0;  //Identifies as packet Id "afcs"
              *buf_ptr++ = 0;
              *buf_ptr++ = 0;
              *buf_ptr++ = 15;
              *buf_ptr++ = 0;
              *buf_ptr++ = sensor[0]/10;
              *buf_ptr++ = 0;
              *buf_ptr++ = sensor[1]/10;
              *buf_ptr++ = 0;
              *buf_ptr++ = sensor[2]/10;
              *buf_ptr++ = 0;
              *buf_ptr++ = sensor[3]/10;
              *buf_ptr++ = 0;
              *buf_ptr++ = sensor[4]/10;
              *buf_ptr++ = 0;
              *buf_ptr++ = 0;
              *buf_ptr++ = 0;
              udp_send( udp_socket, secip, secport, buf, NETWORK_TX_BUFFER_SIZE
                        - UDP_APP_OFFSET, 24 );
                    tcp_tick();
              printf("Sensor Distances ");
              printf("0: %u, 1: %u, 2: %u, 3: %u, 4: %u\n", sensor[0], sensor[1],
                    sensor[2], sensor[3], sensor[4]);
              if ((sensor[2] < 800)&&(sensor[2]>35)){ //obstacle detected by center sensor
//                  if ((sensor[1]<1200)&&(sensor[3]<1200)){
                              wdt_feed();
                              //keep original destination in memory to reinput after obstacle is
passed
                              tcp_tick();
                              orignint = desirednint;
                              origndec = desiredndec;
                              orignsign = desirednsign;
```

```
origeint = desiredeint;
origedec = desirededec;
origesign = desiredesign;
origdint = desireddint;
origddec = desiredddec;
origdsign = desireddsign;
orighdgint = desiredhdgint;
orighdgdec = desiredhdgdec;
orighdgsign = desiredhdgsign;
//stop 7 meters from obstacle, keep sending until recognized by
AFCS
        sensor[2] =  (ad_get( 2 )*1189/1023)+30;
distance = sensor[2]-400;
if (xsign == 0){
        decintadd(orignint, origndec, orignsign, xint, xdec, 1);
}
else{
        decintadd(orignint, origndec, orignsign, xint, xdec, 0);
}
deltanint = genericint;
deltandec = genericdec;
deltansign = genericsign;
decintmult(deltanint, deltandec, 0, deltanint, deltandec, 0);
        xsqrint = genericint;
        xsqrdec = genericdec;
if (ysign == 0){
        decintadd(origeint, origedec, origesign, yint, ydec, 1);
}
else{
        decintadd(origeint, origedec, origesign, yint, ydec, 0);
}
deltaeint = genericint;
deltaedec = genericdec;
deltaesign = genericsign;
decintmult(deltaeint, deltaedec, 0, deltaeint, deltaedec, 0);
ysqrint = genericint;
        ysqrdec = genericdec;
        decintadd(xsqrint, xsqrdec, 0, ysqrint, ysqrdec, 0);
        squareroot(genericint, genericdec);
        prop =
(distance*1000000)/((genericint*100)+(genericdec/1000000));
        propint = prop/1000000;
        propdec = (prop - (propint*1000000))*100;
        decintmult(deltanint, deltandec, 0, propint, propdec, 0);
        decintadd(genericint, genericdec, deltansign, xint, xdec,
xsign);
```

```
                                newnint = genericint;
                                newndec = genericdec;
                                newnsign = genericsign;
                                decintmult(deltaeint, deltaedec, 0, propint, propdec, 0);
                                decintadd(genericint, genericdec, deltaesign, yint, ydec,
ysign);
                                neweint = genericint;
                                newedec = genericdec;
                                newesign = genericsign;

        flyto(16,1,2,newnint,newndec,newnsign,neweint,newedec,newesign,
                                origdint,origddec,origdsign,lookint,lookdec,looksign);
//                              flyto(16,4,2,distance/100,(distance-
((distance/100)*100))*1000000,0,0,0,0,lookint,lookdec,looksign);
                                intransit = 1;
                                printf("stop\n");
                                while(intransit==1){
                                        wdt_feed();
                                        tcp_tick();
                                        for( chan = 0; chan < 5; chan++ ) {
                                   sensor[chan] =  (ad_get( chan )*1189/1023)+30;
                                }
                                buf = buf_ptr = OTCP_TXBUF + UDP_APP_OFFSET;
                                     *buf_ptr++ = 1;  //unimportant time padding
                                     *buf_ptr++ = 2;
                                     *buf_ptr++ = 3;
                                     *buf_ptr++ = 4;
                                     *buf_ptr++ = 9;
                                     *buf_ptr++ = 6;
                                     *buf_ptr++ = 7;
                                     *buf_ptr++ = 8;
                                     *buf_ptr++ = 0;  //Identifies as packet Id "afcs"
                                     *buf_ptr++ = 0;
                                     *buf_ptr++ = 0;
                                     *buf_ptr++ = 15;
                                     *buf_ptr++ = 0;
                                     *buf_ptr++ = sensor[0]/10;
                                     *buf_ptr++ = 0;
                                     *buf_ptr++ = sensor[1]/10;
                                     *buf_ptr++ = 0;
                                     *buf_ptr++ = sensor[2]/10;
                                     *buf_ptr++ = 0;
                                     *buf_ptr++ = sensor[3]/10;
                                     *buf_ptr++ = 0;
                                     *buf_ptr++ = sensor[4]/10;
                                     *buf_ptr++ = 0;
```

```
                                *buf_ptr++ = 0;
                                *buf_ptr++ = 0;
                                udp_send( udp_socket, secip, secport, buf,
NETWORK_TX_BUFFER_SIZE
                                        - UDP_APP_OFFSET, 24 );
                                tcp_tick();
                                printf("ax=%lu, ay=%lu, vx=%lu, vy=%lu,
desirednint=%lu,xint=%lu, desiredeint=%lu, yint=%lu\n",axint, ayint,vxint, vyint,
desirednint, xint, desiredeint, yint);
//                              printf("look = %u %lu.%08lu, psi
= %u %lu.%08lu\n"looksign, lookint, lookdec, psisign, psiint, psidec);

                                if ((axint==0)&&(ayint==0)){
                                        if ((vxint==0)&&(vyint==0)){
                                                if (abs(desirednint-xint)<3){
                                                        if(abs(desiredeint-yint)<3){
//                                                              if ((psiint-
lookint)==0){
//                                                                      if (((psidec-
lookdec)/1000000)==0){

        intransit = 0;
//                                                                      }
//                                                              }
                                                        }
                                                }
                                        }
                                }
                        }


                        printf("wall\n");
//
//                      for( chan = 0; chan < 5; chan++ ) {
//                         sensor[chan] =  (ad_get( chan )*1189/1023)+30;
//                      }
//                              //determine relative plane of the wall
//                              deltalookint = lookint;
//                              deltalookdec = lookdec;
//                              deltalooksign = looksign;
//                              if (sensor[1]!=sensor[3]){
//                                      if (sensor[1]>sensor[3]){
//                                              while ((sensor[1] >
sensor[3])&&(lookint<2)){
//                                                      for (n=0;n<15000000;n++){
//                                                              wdt_feed();
```

```
//                                                      }
//                                                      wdt_feed();
//                                                      diffprev = sensor[1]-sensor[3];
//                                                      sensor[1] =
(ad_get( 1 )*1189/1023)+30;
//                                                      sensor[3] =
(ad_get( 3 )*1189/1023)+30;
//                                                              printf("1  %u  >  3  %u
",sensor[1], sensor[3]);
//                                                              printf("
look %lu.%08lu\n",lookint,lookdec);
//                                                      decintadd(lookint, lookdec, looksign,
0, 10000000, 0);
//                                                      lookint = genericint;
//                                                      lookdec = genericdec;
//                                                      looksign = genericsign;
//                                                      lookat(0, 0, 0, 0, lookint, lookdec,
looksign);
//                                              }
//                                              diff=sensor[3]-sensor[1];
//                                              if (diff>diffprev){
//                                                      decintadd(lookint, lookdec, looksign,
0, 10000000, 1);
//                                                      lookint = genericint;
//                                                      lookdec = genericdec;
//                                                      looksign = genericsign;
//                                                      for (n=0;n<15000000;n++){
//                                                              wdt_feed();
//                                                      }
//                                                      lookat(0, 0, 0, 0, lookint, lookdec,
looksign);
//                                              }
//                                      }
//                                      else {
//                                              while ((sensor[1] <
sensor[3])&&(lookint<2)){
//                                                      for (n=0;n<15000000;n++){
//                                                              wdt_feed();
//                                                      }
//                                                      wdt_feed();
//                                                      diffprev = sensor[3]-sensor[1];
//                                                      sensor[1] =
(ad_get( 1 )*1189/1023)+30;
//                                                      sensor[3] =
(ad_get( 3 )*1189/1023)+30;
```

```
//                                                      printf("1 %u < 3 %u
",sensor[1], sensor[3]);
//                                                      printf("
look %lu.%08lu\n",lookint,lookdec);
//                                                      decintadd(lookint, lookdec, looksign,
0, 10000000, 1);
//                                                      lookint = genericint;
//                                                      lookdec = genericdec;
//                                                      looksign = genericsign;
//                                                      lookat(0, 0, 0, 0, lookint, lookdec,
looksign);
//                                              }
//                                              diff=sensor[1]-sensor[3];
//                                              if (diff>diffprev){
//                                                      decintadd(lookint, lookdec, looksign,
0, 10000000, 0);
//                                                      lookint = genericint;
//                                                      lookdec = genericdec;
//                                                      looksign = genericsign;
//                                                      for (n=0;n<15000000;n++){
//                                                              wdt_feed();
//                                                      }
//                                                      lookat(0, 0, 0, 0, lookint, lookdec,
looksign);
//                                              }
//                                      }
//
//                              }
//                              for (n=0;n<10000000;n++){
//                                      wdt_feed();
//                              }
//                              printf("done");
//                              for (n=0;n<100000000;n++){
//                                      wdt_feed();
//                              }
//                              if (deltalooksign == 0){
//                                      decintadd(lookint, lookdec, looksign, deltalookint,
deltalookdec, 1);
//                              }
//                              else{
//                                      decintadd(lookint, lookdec, looksign, deltalookint,
deltalookdec, 0);
//                              }
//                              deltalooksign = genericsign;
//                              printf("look = %lu %lu.%08lu\n",looksign, lookint,
lookdec);
```

```
//                              if (deltalooksign==1){
//                                      if (looksign == 0){
//                                              decintadd(1, 57079633, 0, lookint, lookdec,
0);
//                                              decintadd(psiint, psidec, psisign, genericint,
genericdec, 0);
//                                      }
//                                      else{
//                                              decintadd(1, 57079633, 0, lookint, lookdec,
1);
//                                              decintadd(psiint, psidec, psisign, genericint,
genericdec, 0);
//                              }
//                              }
//                              else{
//                                      if (looksign == 1){
//                                              decintadd(1, 57079633, 0, lookint, lookdec,
0);
//                                              decintadd(psiint, psidec, psisign, genericint,
genericdec, 1);
//                                      }
//                                      else{
//                                              decintadd(1, 57079633, 0, lookint, lookdec,
1);
//                                              decintadd(psiint, psidec, psisign, genericint,
genericdec, 1);
//                              }
//                              }
//                              printf("hdg = %lu.%08lu\n",genericint, genericdec);
//                              printf("psi = %u %lu.%08lu\n",psisign, psiint, psidec);
//                              psiwantint = genericint;
//                              psiwantdec = genericdec;
//                              psiwantsign = genericsign;
//                              if (psiwantint>2){
//                                      if ((psiwantdec>14159265)||(psiwantint>3)){
//                                              if (psiwantsign == 1){
//                                                      psiwantsign = 0;
//                                              }
//                                              else{
//                                                      psiwantsign = 1;
//                                              }
//                                              decintadd(6, 28318531, 0, psiwantint,
psiwantdec, 1);
//                                              psiwantint = genericint;
//                                              psiwantdec = genericdec;
//                                      }
```

```
//                      }
//                      printf("psiwant = %u %lu.%08lu\n",psiwantsign,
psiwantint, psiwantdec);

                        wallside = 2;
                        while (wallside == 2){
                                wdt_feed();
                                lookat(0, thetaint, thetadec, thetasign, 0, 87266463,
1);
                                for (n=0;n<3000;n++){
                                        printf("feed\n");
                                        wdt_feed();
                                }
                                sensor[0]=(ad_get( 0 )*1189/1023)+30;
                                if (sensor[0] > 1200){
                                        wallside = 4;
                                        flyto(16,1,2,xint,xdec,xsign,yint,
                                                ydec,ysign,zint,zdec,zsign,
                                                1, 57079633, 1);
                                        intransit = 1;
                                        while(intransit==1){
                                                wdt_feed();
                                                tcp_tick();
                                                for( chan = 0; chan < 5; chan++ ) {
                                        sensor[chan] =
(ad_get( chan )*1189/1023)+30;
                                        }
                                        buf = buf_ptr = OTCP_TXBUF +
UDP_APP_OFFSET;
                                            *buf_ptr++ = 1;  //unimportant time
padding
                                            *buf_ptr++ = 2;
                                            *buf_ptr++ = 3;
                                            *buf_ptr++ = 4;
                                            *buf_ptr++ = 9;
                                            *buf_ptr++ = 6;
                                            *buf_ptr++ = 7;
                                            *buf_ptr++ = 8;
                                            *buf_ptr++ = 0;  //Identifies as packet Id
"afcs"
                                            *buf_ptr++ = 0;
                                            *buf_ptr++ = 0;
                                            *buf_ptr++ = 15;
                                            *buf_ptr++ = 0;
                                            *buf_ptr++ = sensor[0]/10;
                                            *buf_ptr++ = 0;
```

```
                *buf_ptr++ = sensor[1]/10;
                *buf_ptr++ = 0;
                *buf_ptr++ = sensor[2]/10;
                *buf_ptr++ = 0;
                *buf_ptr++ = sensor[3]/10;
                *buf_ptr++ = 0;
                *buf_ptr++ = sensor[4]/10;
                *buf_ptr++ = 0;
                *buf_ptr++ = 0;
                *buf_ptr++ = 0;
                udp_send( udp_socket, secip, secport, buf,
NETWORK_TX_BUFFER_SIZE
                        - UDP_APP_OFFSET, 24 );
            tcp_tick();
            if ((psiint-1)==0){
                if (((psidec-
57079633)/1000000)==0){
                        intransit = 0;
                }
            }
        }
        lookat(0, thetaint, thetadec, thetasign, 0,
87266463, 0);
        for (n=0;n<6000;n++){
            printf("feed\n");
            wdt_feed();
        }
    }
    else{
        lookat(0, thetaint, thetadec, thetasign, 0,
87266463, 0);
        for (n=0;n<3000;n++){
            printf("feed\n");
            wdt_feed();
        }
        sensor[4]=(ad_get( 4 )*1189/1023)+30;
        if (sensor[4] > 1200){
            wallside = 0;
            flyto(16,1,2,xint,xdec,xsign,yint,
                ydec,ysign,zint,zdec,zsign,
                1, 57079633, 0);
            intransit = 1;
            while(intransit==1){
                wdt_feed();
                tcp_tick();
```

```
                    for( chan = 0; chan < 5;
chan++ ) {
                  sensor[chan] =
(ad_get( chan )*1189/1023)+30;
                }
                buf = buf_ptr = OTCP_TXBUF +
UDP_APP_OFFSET;
                    *buf_ptr++ = 1;  //unimportant
time padding
                    *buf_ptr++ = 2;
                    *buf_ptr++ = 3;
                    *buf_ptr++ = 4;
                    *buf_ptr++ = 9;
                    *buf_ptr++ = 6;
                    *buf_ptr++ = 7;
                    *buf_ptr++ = 8;
                    *buf_ptr++ = 0;  //Identifies as
packet Id "afcs"
                    *buf_ptr++ = 0;
                    *buf_ptr++ = 0;
                    *buf_ptr++ = 15;
                    *buf_ptr++ = 0;
                    *buf_ptr++ = sensor[0]/10;
                    *buf_ptr++ = 0;
                    *buf_ptr++ = sensor[1]/10;
                    *buf_ptr++ = 0;
                    *buf_ptr++ = sensor[2]/10;
                    *buf_ptr++ = 0;
                    *buf_ptr++ = sensor[3]/10;
                    *buf_ptr++ = 0;
                    *buf_ptr++ = sensor[4]/10;
                    *buf_ptr++ = 0;
                    *buf_ptr++ = 0;
                    *buf_ptr++ = 0;
                    udp_send( udp_socket, secip,
secport, buf, NETWORK_TX_BUFFER_SIZE
                                -
UDP_APP_OFFSET, 24 );
                    tcp_tick();
                    if ((psiint-1)==0){
                        if (((psidec-
57079633)/1000000)==0){
                            intransit = 0;
                        }
                    }
                }
```

```
                                        lookat(0, thetaint, thetadec, thetasign,
0, 87266463, 1);
                                        for (n=0;n<6000;n++){
                                                printf("feed\n");
                                                wdt_feed();
                                        }
                                }
                        }
                        if (wallside == 2){
                                reltoabs(5,0,1,0,0,0);
                                flyto(16,1,2,genericint, genericdec,
genericsign,
                                        genericint2, genericdec2,
genericsign2,
                                        origdint, origddec, origdsign, 0,0,0);
//                              flyto(16,4,2,5,0,1,0,0,0,0,0,0,0,0,0);
                                intransit = 1;
                                printf("back up\n");
                                while(intransit==1){
                                        wdt_feed();
                                        tcp_tick();
                                        for( chan = 0; chan < 5; chan++ ) {
                                sensor[chan] =
(ad_get( chan )*1189/1023)+30;
                                }
                                buf = buf_ptr = OTCP_TXBUF +
UDP_APP_OFFSET;
                                        *buf_ptr++ = 1;  //unimportant time
padding
                                        *buf_ptr++ = 2;
                                        *buf_ptr++ = 3;
                                        *buf_ptr++ = 4;
                                        *buf_ptr++ = 9;
                                        *buf_ptr++ = 6;
                                        *buf_ptr++ = 7;
                                        *buf_ptr++ = 8;
                                        *buf_ptr++ = 0;  //Identifies as packet Id
"afcs"
                                        *buf_ptr++ = 0;
                                        *buf_ptr++ = 0;
                                        *buf_ptr++ = 15;
                                        *buf_ptr++ = 0;
                                        *buf_ptr++ = sensor[0]/10;
                                        *buf_ptr++ = 0;
                                        *buf_ptr++ = sensor[1]/10;
                                        *buf_ptr++ = 0;
```

```
                                        *buf_ptr++ = sensor[2]/10;
                                        *buf_ptr++ = 0;
                                        *buf_ptr++ = sensor[3]/10;
                                        *buf_ptr++ = 0;
                                        *buf_ptr++ = sensor[4]/10;
                                        *buf_ptr++ = 0;
                                        *buf_ptr++ = 0;
                                        *buf_ptr++ = 0;
                                        udp_send( udp_socket, secip, secport, buf,
NETWORK_TX_BUFFER_SIZE
                                                - UDP_APP_OFFSET, 24 );
                                    tcp_tick();
            //                      printf("ax=%lu, ay=%lu, vx=%lu,
vy=%lu, desirednint=%lu,xint=%lu, desiredeint=%lu, yint=%lu\n",axint, ayint,vxint,
vyint, desirednint, xint, desiredeint, yint);
                                    if ((axint==0)&&(ayint==0)){
                                        if
((vxint==0)&&(vyint==0)){
                                            if (abs(desirednint-
xint)<3){

        if(abs(desiredeint-yint)<3){

        intransit=0;
                                                }
                                            }
                                        }
                                    }
                                }
                            }

                    tcp_tick();
                        for( chan = 0; chan < 5; chan++ ) {
                    sensor[chan] =  (ad_get( chan )*1189/1023)+30;
                }
                printf("Sensor Distances ");
                    printf("0: %u, 1: %u, 2: %u, 3: %u, 4: %u\n", sensor[0],
sensor[1],
                        sensor[2], sensor[3], sensor[4]);
                        obstacle = 1;
                        if (wallside == 4){
                            printf("1");
                            while(obstacle == 1){
                                wdt_feed();
                                printf("2");
```

```
                                                obstacle = 0;
                                                if (sensor[0]>1100){
                                                        printf("3");
                                                        if ((sensor[4]<1100)){
                                                                obstacle = 1;
                                                                for( chan = 0; chan < 5;
chan++ ) {
                                                        sensor[chan] =
(ad_get( chan )*1189/1023)+30;
                                                    }
                                                    printf("Sensor Distances ");
                                                         printf("0: %u, 1: %u, 2: %u, 3: %u,
4: %u\n", sensor[0], sensor[1],
                                                                sensor[2], sensor[3],
sensor[4]);
                                                     if (sensor[4]<600){
//                                                          if ((sensor[3]<600)){
                                                                    if (looksign == 1){

        reltoabs(5,0,0,7-(sensor[4]/100),0,0);

        flyto(16,1,2,genericint, genericdec, genericsign,

        genericint2, genericdec2, genericsign2,

        origdint, origddec, origdsign, 0,0,0);
//
        flyto(16,4,2,5,0,0,7-(sensor[4]/100),0,0,0,0,0,0,0,0);
                                                                        tcp_tick();
                                                                }
                                                                else{

        reltoabs(5,0,0,7-(sensor[4]/100),0,1);

        flyto(16,1,2,genericint, genericdec, genericsign,

        genericint2, genericdec2, genericsign2,

        origdint, origddec, origdsign, 0,0,0);
//
        flyto(16,4,2,5,0,0,7-(sensor[4]/100),0,1,0,0,0,0,0,0);
                                                                        tcp_tick();
                                                                }
//                                                          }
                                                     }
                                                     else if (sensor[4]>600){
```

```
//                                                      if ((sensor[3]>600)){
                                                              if (looksign == 1){

        reltoabs(5,0,0,(sensor[4]/100)-5,0,1);

        flyto(16,1,2,genericint, genericdec, genericsign,

        genericint2, genericdec2, genericsign2,

        origdint, origddec, origdsign, 0,0,0);
//
        flyto(16,4,2,5,0,0,(sensor[4]/100)-5,0,1,0,0,0,0,0,0);
                                                                      tcp_tick();
                                                              }
                                                              else{

        reltoabs(5,0,0,(sensor[4]/100)-5,0,0);

        flyto(16,1,2,genericint, genericdec, genericsign,

        genericint2, genericdec2, genericsign2,

        origdint, origddec, origdsign, 0,0,0);
//
        flyto(16,4,2,5,0,0,(sensor[4]/100)-5,0,0,0,0,0,0,0,0);
                                                                      tcp_tick();
                                                              }
//                                                      }
                                                    }
                                                    else{
                                                          reltoabs(5,0,0,0,0,0);
                                                                flyto(16,1,3,genericint,
genericdec, genericsign,
                                                                      genericint2,
genericdec2, genericsign2,
                                                                      origdint,
origddec, origdsign, 0,0,0);
//
        flyto(16,4,3,5,0,0,0,0,0,0,0,0,0,0,0);
                                                          tcp_tick();
                                                    }

                                                intransit =1;
                                                printf("continuing along wall\n");
                                                while(intransit==1){
                                                   wdt_feed();
```

```
                                                    tcp_tick();
                                                    printf("desirednint=%lu, xint=%lu,
desiredeint=%lu,        yint=%lu\n", desirednint, xint, desiredeint, yint);
                                                                for( chan = 0; chan <
5; chan++ ) {
                                                       sensor[chan] =
(ad_get( chan )*1189/1023)+30;
                                                    }
                                                    buf = buf_ptr = OTCP_TXBUF
+ UDP_APP_OFFSET;
                                                        *buf_ptr++ = 1;
//unimportant time padding
                                                        *buf_ptr++ = 2;
                                                        *buf_ptr++ = 3;
                                                        *buf_ptr++ = 4;
                                                        *buf_ptr++ = 9;
                                                        *buf_ptr++ = 6;
                                                        *buf_ptr++ = 7;
                                                        *buf_ptr++ = 8;
                                                        *buf_ptr++ = 0;
//Identifies as packet Id "afcs"
                                                        *buf_ptr++ = 0;
                                                        *buf_ptr++ = 0;
                                                        *buf_ptr++ = 15;
                                                        *buf_ptr++ = 0;
                                                        *buf_ptr++ = sensor[0]/10;
                                                        *buf_ptr++ = 0;
                                                        *buf_ptr++ = sensor[1]/10;
                                                        *buf_ptr++ = 0;
                                                        *buf_ptr++ = sensor[2]/10;
                                                        *buf_ptr++ = 0;
                                                        *buf_ptr++ = sensor[3]/10;
                                                        *buf_ptr++ = 0;
                                                        *buf_ptr++ = sensor[4]/10;
                                                        *buf_ptr++ = 0;
                                                        *buf_ptr++ = 0;
                                                        *buf_ptr++ = 0;
                                                        udp_send( udp_socket,
secip, secport, buf, NETWORK_TX_BUFFER_SIZE
                                                                    -
UDP_APP_OFFSET, 24 );
                                                            tcp_tick();
                                                            if (abs(desirednint-
xint)<3){
                                                                    if
(abs(desiredeint-yint)<3){
```

```
intransit=0;
                                                                    }
                                                                }
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                            else {
                                                printf("4");
                                                while(obstacle == 1){
                                                    wdt_feed();
                                                    printf("5");
                                                    obstacle = 0;
                                                    if (sensor[4]>1100){
                                                        printf("6");
                                                        if ((sensor[0]<1100)){
                                                            obstacle = 1;
                                                            for( chan = 0; chan < 5;
chan++ ) {
                                                        sensor[chan] =
(ad_get( chan )*1189/1023)+30;
                                                    }
                                                    printf("Sensor Distances ");
                                                       printf("0: %u, 1: %u, 2: %u, 3: %u,
4: %u\n", sensor[0], sensor[1],
                                                            sensor[2], sensor[3],
sensor[4]);
                                                    if (sensor[0]<600){
//                                                     if ((sensor[1]<600)){
                                                                if (looksign == 1){

        reltoabs(5,0,0,7-(sensor[0]/100),0,0);

        flyto(16,1,2,genericint, genericdec, genericsign,

        genericint2, genericdec2, genericsign2,

        origdint, origddec, origdsign, 0,0,0);
//
        flyto(16,4,2,5,0,0,7-(sensor[0]/100),0,0,0,0,0,0,0,0);
                                                                    tcp_tick();
                                                                }
                                                                else{
```

```
        reltoabs(5,0,0,7-(sensor[0]/100),0,1);
        flyto(16,1,2,genericint, genericdec, genericsign,
        genericint2, genericdec2, genericsign2,
        origdint, origddec, origdsign, 0,0,0);
//
        flyto(16,4,2,5,0,0,7-(sensor[0]/100),0,1,0,0,0,0,0,0);
                                                                tcp_tick();
                                                        }
//                                                  }
                                                    }
                                                    else if (sensor[0]>600){
//                                                      if ((sensor[1]>600)){
                                                                if (looksign == 1){
        reltoabs(5,0,0,(sensor[0]/100)-5,0,1);
        flyto(16,1,2,genericint, genericdec, genericsign,
        genericint2, genericdec2, genericsign2,
        origdint, origddec, origdsign, 0,0,0);
//
        flyto(16,4,2,5,0,0,(sensor[0]/100)-5,0,1,0,0,0,0,0,0);
                                                                tcp_tick();
                                                        }
                                                        else{
        reltoabs(5,0,0,(sensor[0]/100)-5,0,0);
        flyto(16,1,2,genericint, genericdec, genericsign,
        genericint2, genericdec2, genericsign2,
        origdint, origddec, origdsign, 0,0,0);
//
        flyto(16,4,2,5,0,0,(sensor[0]/100)-5,0,0,0,0,0,0,0,0);
                                                                tcp_tick();
                                                        }
//                                                  }
                                                }
                                                else{
                                                    reltoabs(5,0,0,0,0,0);
```

```
                                                                    flyto(16,1,3,genericint,
genericdec, genericsign,
                                                                        genericint2,
genericdec2, genericsign2,
                                                                        origdint,
origddec, origdsign, 0,0,0);
//
        flyto(16,4,3,5,0,0,0,0,0,0,0,0,0,0,0,0);
                                                                    tcp_tick();
                                                                }
                                                            intransit = 1;
                                                            printf("continuing along wall\n");
                                                            while(intransit==1){
                                                               wdt_feed();
                                                               tcp_tick();
                                                               printf("desirednint=%lu, xint=%lu,
desiredeint=%lu,        yint=%lu\n", desirednint, xint, desiredeint, yint);
                                                                    for( chan = 0; chan <
5; chan++ ) {
                                                                sensor[chan] =
(ad_get( chan )*1189/1023)+30;
                                                            }
                                                            buf = buf_ptr = OTCP_TXBUF
+ UDP_APP_OFFSET;
                                                                *buf_ptr++ = 1;
//unimportant time padding
                                                                *buf_ptr++ = 2;
                                                                *buf_ptr++ = 3;
                                                                *buf_ptr++ = 4;
                                                                *buf_ptr++ = 9;
                                                                *buf_ptr++ = 6;
                                                                *buf_ptr++ = 7;
                                                                *buf_ptr++ = 8;
                                                                *buf_ptr++ = 0;
//Identifies as packet Id "afcs"
                                                                *buf_ptr++ = 0;
                                                                *buf_ptr++ = 0;
                                                                *buf_ptr++ = 15;
                                                                *buf_ptr++ = 0;
                                                                *buf_ptr++ = sensor[0]/10;
                                                                *buf_ptr++ = 0;
                                                                *buf_ptr++ = sensor[1]/10;
                                                                *buf_ptr++ = 0;
                                                                *buf_ptr++ = sensor[2]/10;
                                                                *buf_ptr++ = 0;
                                                                *buf_ptr++ = sensor[3]/10;
```

```
                                                        *buf_ptr++ = 0;
                                                        *buf_ptr++ = sensor[4]/10;
                                                        *buf_ptr++ = 0;
                                                        *buf_ptr++ = 0;
                                                        *buf_ptr++ = 0;
                                                        udp_send( udp_socket,
secip, secport, buf, NETWORK_TX_BUFFER_SIZE
                                                                -
UDP_APP_OFFSET, 24 );
                                                            tcp_tick();
                                                            if (abs(desirednint-
xint)<3){
                                                                if
(abs(desiredeint-yint)<3){

        intransit=0;
                                                                }
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                        reltoabs(5,0,0,0,0,0);
                                        flyto(16,1,3,genericint, genericdec, genericsign,
                                                genericint2, genericdec2, genericsign2,
                                                origdint, origddec, origdsign, 0,0,0);
//                                      flyto(16,4,3,5,0,0,0,0,0,0,0,0,0,0,0);
                                        tcp_tick();
                                intransit = 1;
                                printf("extra\n");
                                while(intransit==1){
                                   wdt_feed();
                                   tcp_tick();
                                   for( chan = 0; chan < 5; chan++ ) {
                                            sensor[chan] =  (ad_get( chan )*1189/1023)+30;
                                       }
                                       buf = buf_ptr = OTCP_TXBUF + UDP_APP_OFFSET;
                                            *buf_ptr++ = 1;  //unimportant time padding
                                            *buf_ptr++ = 2;
                                            *buf_ptr++ = 3;
                                            *buf_ptr++ = 4;
                                            *buf_ptr++ = 9;
                                            *buf_ptr++ = 6;
                                            *buf_ptr++ = 7;
                                            *buf_ptr++ = 8;
```

```
                                        *buf_ptr++ = 0;  //Identifies as packet Id "afcs"
                                        *buf_ptr++ = 0;
                                        *buf_ptr++ = 0;
                                        *buf_ptr++ = 15;
                                        *buf_ptr++ = 0;
                                        *buf_ptr++ = sensor[0]/10;
                                        *buf_ptr++ = 0;
                                        *buf_ptr++ = sensor[1]/10;
                                        *buf_ptr++ = 0;
                                        *buf_ptr++ = sensor[2]/10;
                                        *buf_ptr++ = 0;
                                        *buf_ptr++ = sensor[3]/10;
                                        *buf_ptr++ = 0;
                                        *buf_ptr++ = sensor[4]/10;
                                        *buf_ptr++ = 0;
                                        *buf_ptr++ = 0;
                                        *buf_ptr++ = 0;
                                        udp_send( udp_socket, secip, secport, buf,
NETWORK_TX_BUFFER_SIZE
                                                - UDP_APP_OFFSET, 24 );
                                        tcp_tick();
//                                      printf("desirednint=%lu, xint=%lu, desiredeint=%lu,
        yint=%lu\n", desirednint, xint, desiredeint, yint);
                                        if (abs(desirednint-xint)<3){
                                                if (abs(desiredeint-yint)<3){
                                                        intransit=0;
                                                }
                                        }
                                }
//                                      flyto(16,1,1,xint,xdec,xsign,yint,
//                                                      ydec,ysign,zint,zdec,zsign,
//                                                      orighdgint, orighdgdec,
orighdgsign);
//                                      intransit = 1;
//                                      printf("rotate\n");
//                                      while(intransit==1){
//                                              wdt_feed();
//                                              tcp_tick();
//                                              printf("orighdg=%u %lu.%08lu,
psi=%u %lu.%08lu\n",orighdgsign, orighdgint, orighdgdec,
//                                                      psisign, psiint, psidec);
//                                              if ((psiint-orighdgint)==0){
//                                                      if (((psidec-orighdgdec)/1000000)==0){
//                                                              intransit = 0;
//                                                      }
//                                              }
```

```
//                                      }
                                        flyto(16, 1, 5, orignint, origndec, orignsign, origeint,
origedec,
                                                origesign, origdint, origddec, origdsign, 0,0,0);
                                        tcp_tick();
//                      }
//                      else if (sensor[1]<800){
//                              tcp_tick();
//                              orignint = desirednint;
//                              origndec = desiredndec;
//                              orignsign = desirednsign;
//                              origeint = desiredeint;
//                              origedec = desirededec;
//                              origesign = desiredesign;
//                              origdint = desireddint;
//                              origddec = desiredddec;
//                              origdsign = desireddsign;
//                              flyto(16,4,2,2,0,0,5,0,0,0,0,0,0,0,0);
//                              intransit =1;
//                              printf("obstacle front left\n");
//                            while(intransit==1){
//                              wdt_feed();
////                            printf("desirednint=%lu, xint=%lu, desiredeint=%lu,
        yint=%lu\n", desirednint, xint, desiredeint, yint);
//                                      tcp_tick();
//                                      if (abs(desirednint-xint)<3){
//                                              if (abs(desiredeint-yint)<3){
//                                                      intransit=0;
//                                              }
//                                      }
//                                      }
//                                      printf("front left done\n");
//                                      flyto(16, 1, 3, orignint, origndec, orignsign, origeint,
origedec,
//                                              origesign, origdint, origddec, origdsign, 0,0,0);
//                                      tcp_tick();
//                      }
//                      else if (sensor[3]<800){
//                              tcp_tick();
//                              orignint = desirednint;
//                              origndec = desiredndec;
//                              orignsign = desirednsign;
//                              origeint = desiredeint;
//                              origedec = desirededec;
//                              origesign = desiredesign;
//                              origdint = desireddint;
```

```
//                      origddec = desiredddec;
//                      origdsign = desireddsign;
//                      flyto(16,4,2,2,0,0,5,0,1,0,0,0,0,0,0);
//                      printf("obstacle front right\n");
//                      while(intransit==1){
//                              wdt_feed();
////                    printf("desirednint=%lu, xint=%lu, desiredeint=%lu,
        yint=%lu\n", desirednint, xint, desiredeint, yint);
//                                      tcp_tick();
//                                      if (abs(desirednint-xint)<3){
//                                              if (abs(desiredeint-yint)<3){
//                                                      intransit=0;
//                                              }
//                                      }
//                              }
//                              flyto(16, 1, 3, orignint, origndec, orignsign, origeint,
origedec,
//                                      origesign, origdint, origddec, origdsign, 0,0,0);
//                              tcp_tick();
//              }
//              else if (sensor[2]<800){
//                      tcp_tick();
//                      orignint = desirednint;
//                      origndec = desiredndec;
//                      orignsign = desirednsign;
//                      origeint = desiredeint;
//                      origedec = desirededec;
//                      origesign = desiredesign;
//                      origdint = desireddint;
//                      origddec = desiredddec;
//                      origdsign = desireddsign;
//                      flyto(16,4,2,2,0,0,10,0,0,0,0,0,0,0,0);
//                      printf("obstacle only in front middle\n");
//                      while(intransit==1){
//                              wdt_feed();
////                    printf("desirednint=%lu, xint=%lu, desiredeint=%lu,
        yint=%lu\n", desirednint, xint, desiredeint, yint);
//                                      tcp_tick();
//                                      if (abs(desirednint-xint)<3){
//                                              if (abs(desiredeint-yint)<3){
//                                                      intransit=0;
//                                              }
//                                      }
//                              }
//                              flyto(16, 1, 3, orignint, origndec, orignsign, origeint,
origedec,
```

```
//                                      origesign, origdint, origddec, origdsign, 0,0,0);
//                                tcp_tick();
//                }

                }
                else if (sensor[1]<800){
                        tcp_tick();
                        orignint = desirednint;
                origndec = desiredndec;
                orignsign = desirednsign;
                origeint = desiredeint;
                origedec = desirededec;
                origesign = desiredesign;
                origdint = desireddint;
                origddec = desiredddec;
                origdsign = desireddsign;
                psiwantint = (((lookint*100000000)+lookdec)/15708)/10000;
                psiwantdec = ((((lookint*100000000)+lookdec)/15708)-
(psiwantint*10000))*10000;
                psiwantsign = looksign;
                decintadd(2,14450692,0,psiwantint,psiwantdec,psiwantsign);
                reltoabs(1,0,0,genericint,genericdec,genericsign);
                        flyto(16,1,2,genericint, genericdec, genericsign,
                                genericint2, genericdec2, genericsign2,
                                origdint, origddec, origdsign, 0,0,0);
//                      flyto(16,4,2,1,0,0,genericint,genericdec,genericsign,0,0,0,0,0,0);
                intransit = 1;
                while(intransit==1){
                wdt_feed();
                tcp_tick();
                                printf("desirednint=%lu, xint=%lu, desiredeint=%lu,
        yint=%lu\n", desirednint, xint, desiredeint, yint);
                                for( chan = 0; chan < 5; chan++ ) {
                           sensor[chan] =  (ad_get( chan )*1189/1023)+30;
                        }
                        buf = buf_ptr = OTCP_TXBUF + UDP_APP_OFFSET;
                             *buf_ptr++ = 1;  //unimportant time padding
                             *buf_ptr++ = 2;
                             *buf_ptr++ = 3;
                             *buf_ptr++ = 4;
                             *buf_ptr++ = 9;
                             *buf_ptr++ = 6;
                             *buf_ptr++ = 7;
                             *buf_ptr++ = 8;
                             *buf_ptr++ = 0;  //Identifies as packet Id "afcs"
                             *buf_ptr++ = 0;
```

```
                        *buf_ptr++ = 0;
                        *buf_ptr++ = 15;
                        *buf_ptr++ = 0;
                        *buf_ptr++ = sensor[0]/10;
                        *buf_ptr++ = 0;
                        *buf_ptr++ = sensor[1]/10;
                        *buf_ptr++ = 0;
                        *buf_ptr++ = sensor[2]/10;
                        *buf_ptr++ = 0;
                        *buf_ptr++ = sensor[3]/10;
                        *buf_ptr++ = 0;
                        *buf_ptr++ = sensor[4]/10;
                        *buf_ptr++ = 0;
                        *buf_ptr++ = 0;
                        *buf_ptr++ = 0;
                        udp_send( udp_socket, secip, secport, buf,
NETWORK_TX_BUFFER_SIZE
                                - UDP_APP_OFFSET, 24 );
                        tcp_tick();
                        if (abs(desirednint-xint)<3){
                                if (abs(desiredeint-yint)<3){
                                        intransit=0;
                                }
                        }
                }
                flyto(16, 1, 3, orignint, origndec, orignsign, origeint, origedec,
                        origesign, origdint, origddec, origdsign, 0,0,0);
        }
        else if (sensor[3]<800){
                tcp_tick();
                orignint = desirednint;
        origndec = desiredndec;
        orignsign = desirednsign;
        origeint = desiredeint;
        origedec = desirededec;
        origesign = desiredesign;
        origdint = desireddint;
        origddec = desiredddec;
        origdsign = desireddsign;
                psiwantint = (((lookint*100000000)+lookdec)/15708)/10000;
        psiwantdec = ((((lookint*100000000)+lookdec)/15708)-
(psiwantint*10000))*10000;
        psiwantsign = looksign;
        decintadd(2,14450692,1,psiwantint,psiwantdec,psiwantsign);
        reltoabs(1,0,0,genericint,genericdec,genericsign);
                flyto(16,1,2,genericint, genericdec, genericsign,
```

```
                                genericint2, genericdec2, genericsign2,
                                origdint, origddec, origdsign, 0,0,0);
//                          flyto(16,4,2,1,0,0,genericint,genericdec,genericsign,0,0,0,0,0,0);
                intransit = 1;
                while(intransit==1){
                wdt_feed();
                tcp_tick();
                                printf("desirednint=%lu, xint=%lu, desiredeint=%lu,
        yint=%lu\n", desirednint, xint, desiredeint, yint);
                                for( chan = 0; chan < 5; chan++ ) {
                           sensor[chan] =  (ad_get( chan )*1189/1023)+30;
                        }
                        buf = buf_ptr = OTCP_TXBUF + UDP_APP_OFFSET;
                              *buf_ptr++ = 1;  //unimportant time padding
                              *buf_ptr++ = 2;
                              *buf_ptr++ = 3;
                              *buf_ptr++ = 4;
                              *buf_ptr++ = 9;
                              *buf_ptr++ = 6;
                              *buf_ptr++ = 7;
                              *buf_ptr++ = 8;
                              *buf_ptr++ = 0;  //Identifies as packet Id "afcs"
                              *buf_ptr++ = 0;
                              *buf_ptr++ = 0;
                              *buf_ptr++ = 15;
                              *buf_ptr++ = 0;
                              *buf_ptr++ = sensor[0]/10;
                              *buf_ptr++ = 0;
                              *buf_ptr++ = sensor[1]/10;
                              *buf_ptr++ = 0;
                              *buf_ptr++ = sensor[2]/10;
                              *buf_ptr++ = 0;
                              *buf_ptr++ = sensor[3]/10;
                              *buf_ptr++ = 0;
                              *buf_ptr++ = sensor[4]/10;
                              *buf_ptr++ = 0;
                              *buf_ptr++ = 0;
                              *buf_ptr++ = 0;
                              udp_send( udp_socket, secip, secport, buf,
NETWORK_TX_BUFFER_SIZE
                                        - UDP_APP_OFFSET, 24 );
                              tcp_tick();
                              if (abs(desirednint-xint)<3){
                                        if (abs(desiredeint-yint)<3){
                                                  intransit=0;
                                        }
```

```
                        }
                    }
                    flyto(16, 1, 3, orignint, origndec, orignsign, origeint, origedec,
                        origesign, origdint, origddec, origdsign, 0,0,0);
                }
                else if (sensor[0]<800){
                    tcp_tick();
                    orignint = desirednint;
                origndec = desiredndec;
                orignsign = desirednsign;
                origeint = desiredeint;
                origedec = desirededec;
                origesign = desiredesign;
                origdint = desireddint;
                origddec = desiredddec;
                origdsign = desireddsign;
                    psiwantint = (((lookint*100000000)+lookdec)/15708)/10000;
                psiwantdec = ((((lookint*100000000)+lookdec)/15708)-
(psiwantint*10000))*10000;
                psiwantsign = looksign;
                decintadd(0,83909963,0,psiwantint,psiwantdec,psiwantsign);
                reltoabs(1,0,0,genericint,genericdec,genericsign);
                    flyto(16,1,2,genericint, genericdec, genericsign,
                        genericint2, genericdec2, genericsign2,
                        origdint, origddec, origdsign, 0,0,0);
//                  flyto(16,4,2,1,0,0,genericint,genericdec,genericsign,0,0,0,0,0,0);
                intransit = 1;
                while(intransit==1){
                wdt_feed();
                tcp_tick();
                        printf("desirednint=%lu, xint=%lu, desiredeint=%lu,
        yint=%lu\n", desirednint, xint, desiredeint, yint);
                        for( chan = 0; chan < 5; chan++ ) {
                    sensor[chan] =  (ad_get( chan )*1189/1023)+30;
                    }
                    buf = buf_ptr = OTCP_TXBUF + UDP_APP_OFFSET;
                        *buf_ptr++ = 1;  //unimportant time padding
                        *buf_ptr++ = 2;
                        *buf_ptr++ = 3;
                        *buf_ptr++ = 4;
                        *buf_ptr++ = 9;
                        *buf_ptr++ = 6;
                        *buf_ptr++ = 7;
                        *buf_ptr++ = 8;
                        *buf_ptr++ = 0;  //Identifies as packet Id "afcs"
                        *buf_ptr++ = 0;
```

```
                        *buf_ptr++ = 0;
                        *buf_ptr++ = 15;
                        *buf_ptr++ = 0;
                        *buf_ptr++ = sensor[0]/10;
                        *buf_ptr++ = 0;
                        *buf_ptr++ = sensor[1]/10;
                        *buf_ptr++ = 0;
                        *buf_ptr++ = sensor[2]/10;
                        *buf_ptr++ = 0;
                        *buf_ptr++ = sensor[3]/10;
                        *buf_ptr++ = 0;
                        *buf_ptr++ = sensor[4]/10;
                        *buf_ptr++ = 0;
                        *buf_ptr++ = 0;
                        *buf_ptr++ = 0;
                        udp_send( udp_socket, secip, secport, buf,
NETWORK_TX_BUFFER_SIZE
                                - UDP_APP_OFFSET, 24 );
                        tcp_tick();
                        if (abs(desirednint-xint)<3){
                                if (abs(desiredeint-yint)<3){
                                        intransit=0;
                                }
                        }
                }
                flyto(16, 1, 3, orignint, origndec, orignsign, origeint, origedec,
                        origesign, origdint, origddec, origdsign, 0,0,0);
        }
        else if (sensor[4]<800){
                tcp_tick();
                orignint = desirednint;
        origndec = desirendec;
        orignsign = desirednsign;
        origeint = desiredeint;
        origedec = desirededec;
        origesign = desiredesign;
        origdint = desireddint;
        origddec = desiredddec;
        origdsign = desireddsign;
        psiwantint = (((lookint*100000000)+lookdec)/15708)/10000;
        psiwantdec = ((((lookint*100000000)+lookdec)/15708)-
(psiwantint*10000))*10000;
        psiwantsign = looksign;
        decintadd(0,83909963,1,psiwantint,psiwantdec,psiwantsign);
        reltoabs(1,0,0,genericint,genericdec,genericsign);
                flyto(16,1,2,genericint, genericdec, genericsign,
```

```
                                genericint2, genericdec2, genericsign2,
                                origdint, origddec, origdsign, 0,0,0);
//                          flyto(16,4,2,1,0,0,genericint,genericdec,genericsign,0,0,0,0,0,0);
                intransit = 1;
                while(intransit==1){
                wdt_feed();
                tcp_tick();
                                printf("desirednint=%lu, xint=%lu, desiredeint=%lu,
        yint=%lu\n", desirednint, xint, desiredeint, yint);
                                for( chan = 0; chan < 5; chan++ ) {
                           sensor[chan] =  (ad_get( chan )*1189/1023)+30;
                        }
                        buf = buf_ptr = OTCP_TXBUF + UDP_APP_OFFSET;
                              *buf_ptr++ = 1;  //unimportant time padding
                              *buf_ptr++ = 2;
                              *buf_ptr++ = 3;
                              *buf_ptr++ = 4;
                              *buf_ptr++ = 9;
                              *buf_ptr++ = 6;
                              *buf_ptr++ = 7;
                              *buf_ptr++ = 8;
                              *buf_ptr++ = 0;  //Identifies as packet Id "afcs"
                              *buf_ptr++ = 0;
                              *buf_ptr++ = 0;
                              *buf_ptr++ = 15;
                              *buf_ptr++ = 0;
                              *buf_ptr++ = sensor[0]/10;
                              *buf_ptr++ = 0;
                              *buf_ptr++ = sensor[1]/10;
                              *buf_ptr++ = 0;
                              *buf_ptr++ = sensor[2]/10;
                              *buf_ptr++ = 0;
                              *buf_ptr++ = sensor[3]/10;
                              *buf_ptr++ = 0;
                              *buf_ptr++ = sensor[4]/10;
                              *buf_ptr++ = 0;
                              *buf_ptr++ = 0;
                              *buf_ptr++ = 0;
                              udp_send( udp_socket, secip, secport, buf,
NETWORK_TX_BUFFER_SIZE
                                        - UDP_APP_OFFSET, 24 );
                                tcp_tick();
                                if (abs(desirednint-xint)<3){
                                        if (abs(desiredeint-yint)<3){
                                                intransit=0;
                                        }
```

```
                                }
                        }
                        flyto(16, 1, 3, orignint, orignsec, orignsign, origeint, origedec,
                                origesign, origdint, origddec, origdsign, 0,0,0);
                }
        }
        if( udp_close( udp_socket ) == -1 ) {
                puts( "Error closing UDP socket!\n" ); }
        // Finished with the socket so return to pool
        if( udp_releasesocket( udp_socket ) == -1 ) {
                puts( "Error releasing UDP socket!\n" ); }

}

//called during tcp_tick if data is available to read from buffer
int
        udp_eventlistener( char handle, char event, int ip,
                short port, short buffindex, short datalen )
{
        char buf[1066];
        int i,n;
//      printf("UDP listener\n");
        if( handle != udp_socket ) {
                printf("DNS: not my handle!!!!");
                return( -1 ); }
//      printf("Packet from IP Address: %d.%d.%d.%d\n", IP1(ip),
//              IP2(ip), IP3(ip), IP4(ip) );
        UINT32 id, time, times, temp;
        UINT8 tempbuf[8];

        switch(event) {
                case UDP_EVENT_DATA:
                                RECEIVE_NETWORK_BUF(buf, datalen);
                                id = buf[11];

//                              printf("id = %lu ",id);

                                //get port from ground station to send data
                                if ((id == 1)&&(IP4(ip)!=5)){
                                        secip = ip;
                                        secport = port;

                                }
```

```
                    //stop sending open command when acknowledged by
AFCS
                    if ((id == 2)&&(IP4(ip)==5)){
                            connected = 0;
                    }
                    //autopilot data to get desired position
                    if (id == 14){
                            desired = 0;
                            for (n=0; n<8; n++) {
                               tempbuf[7-n]=buf[n+28];
                            }
                            buftodec (tempbuf);
                            desirednint = genericint;
                            desiredndec = genericdec;
                            desirednsign = genericsign;
                            for (n=0; n<8; n++) {
                               tempbuf[7-n]=buf[n+36];
                            }
                            buftodec (tempbuf);
                            desiredeint = genericint;
                            desirededec = genericdec;
                            desiredesign = genericsign;
                            for (n=0; n<8; n++) {
                               tempbuf[7-n]=buf[n+44];
                            }
                            buftodec (tempbuf);
                            desireddint = genericint;
                            desiredddec = genericdec;
                            desireddsign = genericsign;
                            for (n=0; n<8; n++) {
                               tempbuf[7-n]=buf[n+92];
                            }
                            buftodec (tempbuf);
                            desiredhdgint = genericint;
                            desiredhdgdec = genericdec;
                            desiredhdgsign = genericsign;
                    }
//                  //desired_pos data
//                  if (id == 20){
//                          desired = 0;
//                          for (n=0; n<8; n++) {
//                             tempbuf[7-n]=buf[n+12];
//                          }
//                          buftodec (tempbuf);
//                          desirednint = genericint;
//                          desiredndec = genericdec;
```

```
//                              desirednsign = genericsign;
//                              for (n=0; n<8; n++) {
//                                 tempbuf[7-n]=buf[n+20];
//                              }
//                              buftodec (tempbuf);
//                              desiredeint = genericint;
//                              desirededec = genericdec;
//                              desiredesign = genericsign;
//                              for (n=0; n<8; n++) {
//                                 tempbuf[7-n]=buf[n+28];
//                              }
//                              buftodec (tempbuf);
//                              desireddint = genericint;
//                              desiredddec = genericdec;
//                              desireddsign = genericsign;
//                              for (n=0; n<8; n++) {
//                                 tempbuf[7-n]=buf[n+36];
//                              }
//                              buftodec (tempbuf);
//                              desiredhdgint = genericint;
//                              desiredhdgdec = genericdec;
//                              desiredhdgsign = genericsign;
//                      }

                        //gps data
//                      if (id == 12) {
////                            printf("time = %u\n", buf[3]);
//                            numsat = buf[13];
//                            vdop = buf[15];
//                            for (n=0; n<8; n++) {
//                               tempbuf[7-n]=buf[n+20];
//                            }
//                            buftodec (tempbuf);
//                            vel_nint = genericint;
//                            vel_ndec = genericdec;
//                            vel_nsign = genericsign;
//                            printf("vel_n = ");
//                            if (vel_nsign == 1){
//                                    printf("-");
//                            }
//                            printf("%lu.%08lu ",vel_nint, vel_ndec);
//                            for (n=0; n<8; n++) {
//                               tempbuf[7-n]=buf[n+28];
//                            }
//                            buftodec (tempbuf);
//                            vel_eint = genericint;
```

```
//                                      vel_edec = genericdec;
//                                      vel_esign = genericsign;
//                                      printf("vel_e = ");
//                                      if (vel_esign == 1){
//                                              printf("-");
//                                      }
//                                      printf("%lu.%08lu ",vel_eint, vel_edec);
//                                      for (n=0; n<8; n++) {
//                                         tempbuf[7-n]=buf[n+36];
//                                      }
//                                      buftodec (tempbuf);
//                                      vel_dint = genericint;
//                                      vel_ddec = genericdec;
//                                      vel_dsign = genericsign;
//                                      printf("vel_d = ");
//                                      if (vel_dsign == 1){
//                                              printf("-");
//                                      }
//                                      printf("%lu.%08lu\n",vel_dint, vel_ddec);
//                              }
                                //state packet
                                if (id == 10) {
//                                      for (n=0; n<8; n++) {
//                                         tempbuf[7-n]=buf[n+12];
//                                      }
//                                      buftodec (tempbuf);
//                                      phiint = genericint;
//                                      phidec = genericdec;
//                                      phisign = genericsign;
//                                      printf("phi = ");
//                                      if (phisign == 1){
//                                              printf("-");
//                                      }
//                                      printf("%lu.%08lu ",phiint, phidec);
                                        for (n=0; n<8; n++) {
                                           tempbuf[7-n]=buf[n+20];
                                        }
                                        buftodec (tempbuf);
                                        thetaint = genericint;
                                        thetadec = genericdec;
                                        thetasign = genericsign;
//                                      printf("theta = ");
//                                      if (thetasign == 1){
//                                              printf("-");
//                                      }
//                                      printf("%lu.%08lu ",thetaint, thetadec);
```

```
                        for (n=0; n<8; n++) {
                           tempbuf[7-n]=buf[n+28];
                        }
                        buftodec (tempbuf);
                        psiint = genericint;
                        psidec = genericdec;
                        psisign = genericsign;
//                      printf("psi = ");
//                      if (psisign == 1){
//                              printf("-");
//                      }
//                      printf("%lu.%08lu ",psiint, psidec);
                        for (n=0; n<8; n++) {
                           tempbuf[7-n]=buf[n+60];
                        }
                        buftodec (tempbuf);
                        xint = genericint;
                        xdec = genericdec;
                        xsign = genericsign;
//                      printf("x = ");
//                      if (xsign == 1){
//                              printf("-");
//                      }
//                      printf("%lu.%08lu ",xint, xdec);
                        for (n=0; n<8; n++) {
                           tempbuf[7-n]=buf[n+68];
                        }
                        buftodec (tempbuf);
                        yint = genericint;
                        ydec = genericdec;
                        ysign = genericsign;
//                      printf("y = ");
//                      if (ysign == 1){
//                              printf("-");
//                      }
//                      printf("%lu.%08lu ",yint, ydec);
                        for (n=0; n<8; n++) {
                           tempbuf[7-n]=buf[n+76];
                        }
                        buftodec (tempbuf);
                        zint = genericint;
                        zdec = genericdec;
                        zsign = genericsign;
//                      printf("z = ");
//                      if (zsign == 1){
//                              printf("-");
```

```
//                              }
//                              printf("%lu.%08lu\n",zint, zdec);
                                for (n=0; n<8; n++) {
                                   tempbuf[7-n]=buf[n+84];
                                }
                                buftodec (tempbuf);
                                vxint = genericint;
                                vxdec = genericdec;
                                vxsign = genericsign;
//                              printf("vx = ");
//                              if (vxsign == 1){
//                                      printf("-");
//                              }
//                              printf("%lu.%08lu ",vxint, vxdec);
                                for (n=0; n<8; n++) {
                                   tempbuf[7-n]=buf[n+92];
                                }
                                buftodec (tempbuf);
                                vyint = genericint;
                                vydec = genericdec;
                                vysign = genericsign;
//                              printf("vy = ");
//                              if (vysign == 1){
//                                      printf("-");
//                              }
//                              printf("%lu.%08lu ",vyint, vydec);
                                for (n=0; n<8; n++) {
                                   tempbuf[7-n]=buf[n+100];
                                }
                                buftodec (tempbuf);
                                vzint = genericint;
                                vzdec = genericdec;
                                vzsign = genericsign;
//                              printf("vz = ");
//                              if (vzsign == 1){
//                                      printf("-");
//                              }
//                              printf("%lu.%08lu\n",vzint, vzdec);
                                for (n=0; n<8; n++) {
                                   tempbuf[7-n]=buf[n+132];
                                }
                                buftodec (tempbuf);
                                axint = genericint;
                                for (n=0; n<8; n++) {
                                   tempbuf[7-n]=buf[n+140];
                                }
```

```
                                        buftodec (tempbuf);
                                        ayint = genericint;
                                }
                                break;
                default:
                                puts( "Unknown UDP event" );
        }

        return( 0 );
 }
int reltoabs(UINT32 oneint, UINT32 onedec, UINT32 onesign, UINT32 twoint,
                 UINT32 twodec, UINT32 twosign)
{
        UINT32 angleint, tempxint, tempyint;
        UINT32 angledec, tempxdec, tempydec;
        UINT8  anglesign, xdiffsign, ydiffsign, xdiffsign2, ydiffsign2, tempxsign,
tempysign;

        angleint = psiint;
        angledec = psidec;
        anglesign = psisign;
//      printf("angle =%u %lu.%08lu  loc = %u\n",anglesign, angleint, angledec, loc);
//      if (angleint>2){
//              if ((angledec>14159265)||(angleint>3)){
//                      if (anglesign == 1){
//                              anglesign = 0;
//                      }
//                      else{
//                              anglesign = 1;
//                      }
//                      decintadd(6, 28318531, 0, angleint, angledec, 1);
//                      angleint = genericint;
//                      angledec = genericdec;
//              }
//      }
        if (angleint>0){
                if ((angledec>57079633)||(angleint>1)){
                        if (anglesign == 1){
                                xdiffsign = 1;
                                ydiffsign = 1;
                                xdiffsign2 = 0;
                                ydiffsign2 = 1;
                        }
                        else{
                                xdiffsign = 1;
                                ydiffsign = 0;
```

```
                        xdiffsign2 = 1;
                        ydiffsign2 = 1;
                }
                decintadd(3, 14159265, 0, angleint, angledec, 1);
                angleint = genericint;
                angledec = genericdec;
        }
        else{
                if (anglesign == 1){
                        xdiffsign = 0;
                        ydiffsign = 1;
                        xdiffsign2 = 0;
                        ydiffsign2 = 0;
                }
                else{
                        xdiffsign = 0;
                        ydiffsign = 0;
                        xdiffsign2 = 0;
                        ydiffsign2 = 1;
                }
        }
}
else{
        if (anglesign == 1){
                xdiffsign = 0;
                ydiffsign = 1;
                xdiffsign2 = 0;
                ydiffsign2 = 0;
        }
        else{
                xdiffsign = 0;
                ydiffsign = 0;
                xdiffsign2 = 0;
                ydiffsign2 = 1;
        }
}

//      printf("angle =%u %lu.%08lu, xdiffsign=%u, ydiffsign=%u\n",anglesign,
angleint, angledec, xdiffsign, ydiffsign);

        decintmult(angleint, angledec, 0, 0, 63661977, 0);
        angleint = genericint;
        angledec = genericdec;

        decintmult(oneint, onedec, 0, angleint, angledec, 0);
        tempyint = genericint;
```

```
        tempydec = genericdec;
        decintadd(1, 0, 0, angleint, angledec, 1);
        decintmult(oneint, onedec, 0, genericint, genericdec, 0);
        tempxint = genericint;
        tempxdec = genericdec;
        decintadd(1, 0, 0, angleint, angledec, 1);
        decintmult(twoint, twodec, 0, genericint, genericdec, 0);
        decintadd(genericint, genericdec, ydiffsign2, tempyint, tempydec, ydiffsign);
        tempyint = genericint;
        tempydec = genericdec;
        tempysign = genericsign;
        decintmult(twoint, twodec, 0, angleint, angledec, 0);
        decintadd(genericint, genericdec, xdiffsign2, tempxint, tempxdec, xdiffsign);
        tempxint = genericint;
        tempxdec = genericdec;
        tempxsign = genericsign;
        decintadd(tempyint, tempydec, tempysign, yint, ydec, ysign);
        genericint2 = genericint;
        genericdec2 = genericdec;
        genericsign2 = genericsign;
        decintadd(tempxint, tempxdec, tempxsign, xint, xdec, xsign);
        return(0);
}

long buftodec (UINT8 guess[8])
{
        UINT16 e;
        UINT32 fint, fdec;
        UINT32 etempint, etempdec;
        UINT8 guessbin[64];
        UINT16 divider;
        int n,m;

        for (n=0; n<8; n++) {
           divider=1;
           //printf("%d ", guess[n]);
           for (m=0; m<8; m++) {
                 divider *= 2;
                 guessbin[(8*n)+m]=(guess[n])/(256/divider);
                 //printf("%d",guessbin[(8*n)+m]);
                 guess[n] -=(256/divider)*guessbin[(8*n)+m];
           }
           //printf(" ");
        }
        //printf("\n");
        divider = 1;
```

```
e=0;
for (n=1; n<12; n++){
	divider *= 2;
	e += guessbin[n]*(2048/divider);
}
//printf("e %u ", e);
fdec=0;
divider = 1;
for (n=12; n<27; n++){
	divider *= 2;
	fdec += (100000000*guessbin[n])/divider;
}
//printf("fdec %ld ",fdec);
fint = 1;
if (e == 0){
	if (fdec == 0){
		fint = 0;
	}
}
//printf("fint %ld ",fint);
etempint = 1;
etempdec = 0;
if (e > 1023) {
	if ( e < 1053){
		for (n=0; n<(e-1023); n++){
			etempint *= 2;
		}
	}
	else {
		etempint = 1932735282; //max possible w/o error
	}
}
else if (e < 1023) {
	etempint = 0;
	etempdec = 100000000;
	for (n=0; n<(1023-e); n++){
		etempdec /= 2;
	}
}
//printf("etempint %ld etempdec %ld ",etempint, etempdec);

decintmult(etempint, etempdec, 0 , fint, fdec, 0);

genericsign = 0;
if (guessbin[0] == 1) {
	genericsign = 1;
```

```
	}
	//printf("decimal %ld.%08ld\n",genericint, genericdec);
	return(0);
}



int decintmult(UINT32 oneint, UINT32 onedec, UINT32 onesign, UINT32 twoint,
		UINT32 twodec, UINT32 twosign)
{
	UINT32 fronthalfone, backhalfone, fronthalftwo, backhalftwo;

	fronthalfone = onedec/10000;
	backhalfone = onedec-(fronthalfone*10000);
	fronthalftwo = twodec/10000;
	backhalftwo = twodec-(fronthalftwo*10000);
	genericdec = (fronthalfone*fronthalftwo)+((fronthalfone*backhalftwo)/10000)
		+((fronthalftwo*backhalfone)/10000);

	genericint = oneint*twoint;
	genericint += (oneint*fronthalftwo)/10000;
	genericint += (twoint*fronthalfone)/10000;
	genericdec += ((oneint*fronthalftwo)-(((oneint*fronthalftwo)/10000)*10000))
		*10000;
	genericdec += ((twoint*fronthalfone)-(((twoint*fronthalfone)/10000)*10000))
		*10000;
	genericdec += (oneint*backhalftwo);
	genericdec += (twoint*backhalfone);
	genericint += genericdec / 100000000;
	genericdec -= (genericdec / 100000000)*100000000;
	if ((onesign == 0)&&(twosign==0)){
		genericsign = 0;
	}
	else if ((onesign == 1)&&(twosign==1)){
		genericsign = 0;
	}
	else{
		genericsign = 1;
	}
	return(0);
}

int squareroot(unsigned long oneint, unsigned long onedec)
{
	UINT32 sqrtint, sqrtdec, tempint, tempdec;
	UINT32 adder;
```

```
	UINT8  under;
	short n;

	sqrtint = 0;
	sqrtdec = 0;
	adder = 1000;
	for (n=0; n < 4; n++){
		tempint = 0;
		while (tempint <= oneint){
			sqrtint += adder;
			decintmult(sqrtint, sqrtdec, 0, sqrtint, sqrtdec, 0);
			tempint = genericint;
			tempdec = genericdec;
		}
		sqrtint -= adder;
		adder /= 10;
	}
	adder = 10000000;
	for (n=0; n < 8; n++){
		tempdec = 0;
		tempint = 0;
		under = 1;
		while (under == 1){
			under = 0;
			if (tempint <= oneint){
				if ((tempdec <= onedec)||(tempint!=oneint)){
					sqrtdec += adder;
					decintmult(sqrtint, sqrtdec, 0, sqrtint, sqrtdec, 0);
					tempint = genericint;
					tempdec = genericdec;
					under = 1;
				}
			}
		}
		sqrtdec -= adder;
		adder /= 10;
	}
	genericint = sqrtint;
	genericdec = sqrtdec;


	return(0);
}

int decintadd(UINT32 oneint, UINT32 onedec, UINT32 onesign, UINT32 twoint,
		UINT32 twodec, UINT32 twosign)
```

```
{

        if (twosign==0){
                if (onesign==1){
                        if (oneint>=twoint){
                                oneint -= twoint;
                        }
                        else{
                                oneint = twoint - oneint;
                                onesign = 0;
                        }
                }
                else{
                        oneint += twoint;
                }
                if (onesign==1){
                if (onedec>=twodec){
                        onedec -= twodec;
                }
                else if (oneint>0){
                                onedec += (100000000-twodec);
                                oneint -= 1;
                }
                        else{
                                onedec = twodec-onedec;
                                onesign = 0;
                }
                }
                else{
                        if (onedec<(100000000-twodec)){
                        onedec += twodec;
                }
                else if (onedec==(100000000-twodec)){
                                onedec = 0;
                                oneint += 1;
                }
                        else{
                                onedec -= (100000000-twodec);
                                oneint += 1;
                }
                }
        }
        else{
                if (onesign==0){
                        if (oneint>=twoint){
```

```
                        oneint -= twoint;
                }
                else{
                        oneint = twoint - oneint;
                        onesign = 1;
                }
        }
        else{
                oneint += twoint;
        }
        if (onesign==0){
        if (onedec>=twodec){
                onedec -= twodec;
        }
        else if (oneint>0){
                        onedec += (100000000-twodec);
                        oneint -= 1;
        }
                else{
                        onedec = twodec-onedec;
                        onesign = 1;
        }
        }
        else{
                if (onedec<(100000000-twodec)){
                onedec += twodec;
        }
        else if (onedec==(100000000-twodec)){
                        onedec = 0;
                        oneint += 1;
        }
                else{
                        onedec -= (100000000-twodec);
                        oneint += 1;
        }
        }
    }
    genericsign = onesign;
    genericint = oneint;
    genericdec = onedec;
    return(0);
}
int dectobuf(UINT32 oneint, UINT32 onedec, UINT32 onesign)
{
    UINT16 e;
    UINT32 dectemp, etemp;
```

```
UINT32 inttemp, ftemp, fdectemp, finttemp;
UINT8 guessbin[64];
UINT16 divider;

int n,m;


e = 1023;
inttemp = 1;
dectemp = 0;
if (oneint > 1){
        while (oneint >= inttemp){
                e += 1;
                inttemp *= 2;
        }
        inttemp /= 2;
        e -= 1;
}
else if (oneint == 1){
        e = 1023;
}
else if ((oneint == 0)&&(onedec!=0)){
        inttemp = 0;
        dectemp = 100000000;
        while (onedec < dectemp){
                e -= 1;
                dectemp /= 2;
        }
}
else{
        e = 0;
}
divider = 1;
guessbin[0] = onesign;
etemp = e;
for (n=1; n<12; n++){
        divider *= 2;
guessbin[n]=e /(2048/divider);
e -=(2048/divider)*guessbin[n];
}
fdectemp = 0;
finttemp = 1;
divider = 1;
for (n=12; n<27; n++){
        divider *= 2;
        fdectemp += 100000000/divider;
```

```
                decintmult(inttemp, dectemp, 0, finttemp, fdectemp, 0);
//              printf("%lu.%04lu*%lu.%04lu = %lu.%08lu\n",inttemp, dectemp,
finttemp,
//                      fdectemp, genericint, genericdec);
//              printf("dec %lu\n",100000000/divider);
                if (genericint>oneint){
                        guessbin[n] = 0;
                        fdectemp -= 100000000/divider;
                }
                else if ((genericint==oneint)&&(genericdec>onedec)){
                        guessbin[n] = 0;
                        fdectemp -= 100000000/divider;
                }
                else{
                        guessbin[n] = 1;
//                      printf("used");
                }
        }

        for (n=27; n<64; n++){
                guessbin[n] = 0;
        }
        for (n=0; n<64; n++){
                //printf("%d",guessbin[n]);
                if (((n+1)%8)==0){
                        //printf(" ");
                }
        }
        //printf("\n");
        for (n=0; n<8; n++){
                divider = 1;
                bytbuf[7-n]=0;
                for (m=0; m<8; m++){
                        divider *= 2;
                        bytbuf[7-n]+=guessbin[(n*8)+m]*(256/divider);
                }
                //printf("bytbuf %u n %d\n", bytbuf[n], n);
        }
//      printf("%lu.%04lu = %u*1.%1u%1u%1u%1u%1u%1u\n", oneint, onedec, etemp,
guessbin[12],
//              guessbin[13], guessbin[14], guessbin[15], guessbin[16], guessbin[17]);
        return(0);
}
int flyto(unsigned char failsafe, unsigned char posmode, unsigned char hdgmode,
        unsigned long nint, unsigned long ndec, unsigned char nsign,
        unsigned long eint, unsigned long edec, unsigned char esign,
```

```
        unsigned long dint, unsigned long ddec, unsigned char dsign,
        unsigned long hdgint, unsigned long hdgdec, unsigned char hdgsign)
{
        UINT8 *buf_ptr;
   UINT8 *buf;
   char traveltime1, traveltime2;
   UINT32 desirednintprev, desiredeintprev, desiredndecprev, desirededecprev;

//   random += 1;
//   if (random == 200){
//        random = 0;
//   }

   printf("flyto\n");

   //define flyto packet and send
   buf = buf_ptr = OTCP_TXBUF + UDP_APP_OFFSET;
   int n;
   for (n=0;n<7;n++){  //unimportant time padding
        *buf_ptr++ = 0;
   }
   *buf_ptr++ = 0; //random time change to prevent rejection of duplicate packets
   *buf_ptr++ = 0; //flyto command
   *buf_ptr++ = 0;
   *buf_ptr++ = 0;
   *buf_ptr++ = failsafe; //Identifies as packet Id "flyto" or "failsafe"
   *buf_ptr++ = 0;  //pos_mode
   *buf_ptr++ = posmode;
   *buf_ptr++ = 0;  //hdg_mode
   *buf_ptr++ = hdgmode;
   traveltime1 = -100>>8;
   traveltime2 = -100-(traveltime1<<8);
   *buf_ptr++ = traveltime1;  //transit_time in binary for -500 which means 5 m/s travel
speed
   *buf_ptr++ = traveltime2;
   *buf_ptr++ = 0; // padding/timeout time for failsafe
        *buf_ptr++ = 1;
   dectobuf(nint, ndec, nsign);
   for (n=0;n<8;n++){  //x
        *buf_ptr++ = bytbuf[n];
   }
   dectobuf(eint, edec, esign);
   for (n=0;n<8;n++){  //y
        *buf_ptr++ = bytbuf[n];
   }
   dectobuf(dint, ddec, dsign);
```

```
   for (n=0;n<8;n++){  //z
                *buf_ptr++ = bytbuf[n];
        }
        dectobuf(hdgint, hdgdec, hdgsign);
   for (n=0;n<8;n++){  //hdg
        *buf_ptr++ = bytbuf[n];
   }
   desired = 1;
   while (desired == 1){
        wdt_feed();
        tcp_tick();
   }
   desirednintprev = desirednint;
   desiredeintprev = desiredeint;
   desiredndecprev = desiredndec;
   desirededecprev = desirededec;

        udp_send( udp_socket, remote_ip, 2002, buf, NETWORK_TX_BUFFER_SIZE -
UDP_APP_OFFSET, 52 );
   udp_send( udp_socket, secip, secport, buf, NETWORK_TX_BUFFER_SIZE -
UDP_APP_OFFSET, 52 );
        desired = 1;
   while (desired == 1){
        wdt_feed();
        tcp_tick();
        desired = 1;
                if (desirednintprev != desirednint){
                        desired = 0;
                }
                if (desiredeintprev != desiredeint){
                        desired = 0;
                }
                if (desiredndecprev != desiredndec){
                        desired = 0;
                }
                if (desirededecprev != desirededec){
                        desired = 0;
                }
   }

        return(0);
}
int lookat(UINT8 lookmode, UINT32 pitchint, UINT32 pitchdec, UINT8 pitchsign,
        UINT32 yawint, UINT32 yawdec, UINT8 yawsign)
{
        UINT8 *buf_ptr;
```

```
    UINT8 *buf;

//    random += 1;
//    if (random == 200){
//        random = 0;
//    }


        //define platform packet and send
    buf_ptr = 0;
    buf = buf_ptr = OTCP_TXBUF + UDP_APP_OFFSET;
    int n;
    for (n=0;n<7;n++){  //unimportant time padding
        *buf_ptr++ = 0;
    }
    *buf_ptr++ = 0; //random time change to prevent rejection of duplicate packets
    *buf_ptr++ = 0; //platform command
    *buf_ptr++ = 0;
    *buf_ptr++ = 0;
    *buf_ptr++ = 19;
    *buf_ptr++ = 0;  //mode
    *buf_ptr++ = lookmode;
    for (n=14;n<28;n++){  //padding, relative_roll
        *buf_ptr++ = 0;
    }
    dectobuf(pitchint, pitchdec, pitchsign);
        for (n=0;n<8;n++){  //relative_pitch
        *buf_ptr++ = bytbuf[n];
    }
    dectobuf(yawint, yawdec, yawsign);
        for (n=0;n<8;n++){  //relative_yaw
                *buf_ptr++ = bytbuf[n];
        }
        for (n=0;n<8;n++){  //absolute_roll
        *buf_ptr++ = 0;
    }
    dectobuf(pitchint, pitchdec, pitchsign);
        for (n=0;n<8;n++){  //absolute_pitch
        *buf_ptr++ = bytbuf[n];
    }
    dectobuf(yawint, yawdec, yawsign);
        for (n=0;n<8;n++){  //absolute_yaw
                *buf_ptr++ = bytbuf[n];
        }
        for (n=68;n<92;n++){  //n, e, d
                *buf_ptr++ = 0;
```

```
}
udp_send( udp_socket, remote_ip, 2002, buf, NETWORK_TX_BUFFER_SIZE
        - UDP_APP_OFFSET, 92 );
tcp_tick();
udp_send( udp_socket, secip, secport, buf, NETWORK_TX_BUFFER_SIZE
        - UDP_APP_OFFSET, 92 );
tcp_tick();
return(0);
```

}

VITA

Name: Christopher Isaac Mentzer

Address: Southwest Research Institute
Division 09
6220 Culebra Road
P.O. Drawer 28510
San Antonio, Texas 78229-0519

Email Address: chrismentzer@hotmail.com

Education: B.S., Mechanical Engineering, Texas A&M University, 2002

Professional Experience:

8/02-12/02	Texas A&M University Aerosol Technology Laboratory, College Station, TX Engineering Research Associate
5/02-8/02	Los Alamos National Laboratory, Los Alamos, NM Graduate Research Assistant
1/01-5/01	The 3M Company, St. Paul, MN Co-op Student
5/00-8/00	The 3M Company, Austin, TX Co-op Student
8/99-12/99	The 3M Company, Austin, TX Co-op Student