ENEL589: Fourth Year Engineering Design Project - Part II

Report #1

Department of Electrical and Computer Engineering
University of Calgary, Winter 2012

Project Title: **Autonomous UAV Terrain Avoidance System**

Team Information	
Team number	**30**
Team name (*optional*)	Team Lunch
Team leader	**James Thorne**
2nd Member	Adam Dickin
3rd Member	Rami Abou Ghanem
4th Member	Jennifer Patterson
5th Member	
Customer Information	
Customer (*Full name of the organization if applicable*)	**Lockheed Martin Canada CDL Systems**
Website (*if applicable*)	http://www.cdlsystems.com/
Contact person	Greg Wilding
Contact address	Harvest Hills Office Park Building 5000, #5301, 333 - 96th Ave NE Calgary, AB Canada T3K 0S3
Is there an IP issue with this project?	**Yes**
Is this team in a legal agreement on an IP issue?	**Yes**

Table of Contents

Introduction

Over the past few years, there has been a substantial increase in drones being used for military and police operations. Many of these drones do not have any system in place for collision detection and avoidance besides notifying the operator so that they can change course. This system works well when doing missions in a large open environment, but is insufficient for small, highly maneuverable drones in enclosed or cluttered spaces.

Operator reaction times are too slow to successfully avoid collision, especially when considering the latency introduced by a wireless control harness. Additionally, many of the commercial systems available do not focus on being affordable, but rather on including every feature possible. This results in a drone that is equipped for many tasks, but costs $50,000 or more. This pricing makes drone adoption difficult for casual users, and leaves little room for repair or replacement of drones involved in collisions.

There has been some improvement in this area over the past few years. In fact, some UAVs are now available to the general consumer for around $400. These drones also do not have any kind of collision avoidance, and limited control systems, and are therefore as easy to crash as their high-end counterparts. Some improvement must be made in this area for both the consumer market and the commercial market - ideally, both users would be able to use their drones safely and effectively knowing that systems are in place to avoid crashes.

Our goal as Team Lunch is to solve these problems, and to prove that every autonomous drone can be equipped with our solution. Our project will benefit our customer, Lockheed Martin Canada CDL Systems Ltd., by demonstrating how such a system can be implemented cheaply with off-the-shelf technology available to the general consumer. Our project will also benefit the rest of the UAV market, because we will prove that such features are possible at a low cost. We hope that with time, they will become standard on every UAV. Such an adoption will greatly increase the safety of drones, not only by reducing the risk of a costly crash, but also by reducing the risk of injury to persons and damage to property.

We have decided to focus on taking a commercially available, off the shelf consumer UAV, and equipping it with an autonomous collision detection and avoidance system. This system will allow us to prove that automated accident avoidance is possible during operation of a drone, with minimal operator skill required. Our drone, once outfitted with these systems, will allow unskilled operators to purchase and fly the UAV in close-quarter environments with minimal risk, training, and possibility of damage to the UAV and surrounding property. Hopefully, the outcome of our project will convince drone manufacturers to adopt the use of a collision avoidance and detection system in production model aircraft. We also hope that with the increased availability and lower cost of these autonomous drones, many more organizations will decide to use drones in their everyday operations.

Product Design Specifications

Expected Performance

The product should allow the UAV in use to avoid obstacles in flight. An obstacle is defined as any solid object within the UAV's trajectory. Upon detecting an obstacle, the system will notify the user and either stop the UAV or move away from the obstacle, as required. As of this iteration, the UAV will not be able to reliably avoid wires or meshes, such as chain link fences and nets. The onboard equipment should be able to handle North American outdoor temperatures.

Specifications Related To Performance

The Parrot AR.Drone itself is capable of a speed of 5 m/s, or 18 km/h. It has a weight of 0.9 pounds with the outer hull attached. Flying time on a single battery is about 12 minutes. Based on this speed, we expect the UAV to be able to avoid obstacles which are stationary, or which approach at a speed up to 5 m/s.

According to manufacturer specifications of the sensors, we are capable of sensing obstacles as close as 152.4 mm, although items closer than this will also register as 152.4 mm by the sensor. The sensor can detect obstacles up to 6.5 meters away. Distance measurement resolution is 25.4 mm.

Feeling skeptical of the manufacturer's claims for the ultrasonic sensors, we collected our own distance detection data. In particular, we were interested in finding the angle of the field of view of one of the sensors. Our test procedure is outlined in the Hardware Test Plan section. We found that the ultrasound sensors have a field of view of 40 degrees. This is just short of the 45 degrees we hoped for. Additional ultrasonic sensors will be added on to the system to cover the blind spots. We also found that the size of the obstacle had an effect on its maximum distance for detection. Narrow objects, such as poles and large wires were detectable at 120 cm. Small obstacles of approximately 1 square foot were detectable at 300 cm. Larger barriers of approximately 2 square meters were detectable at 350 cm. Finally, walls were detectable at 600 cm. Since measured results from the sensors were not as effective as the manufacturer claims, the measured values will be our expected specifications.

Measured system response latency was approximately 0.01 seconds. This latency can be considered as negligible, since if an object approached fast enough to collide with the UAV before it has time to respond, the UAV could not possibly have flown out of the way in time.

Finally, the sensor and equipment manufacturer specifications allow the product to be used in temperatures ranging from -40 to 65 degrees Celsius. This is an acceptable outdoor range for most places on Earth.

Low Level Design

Overview of System Components

Our system consists of four major components: the end user, the Android application they use to control the quadcopter, the embedded collision avoidance autopilot software, and the Quadcopter hardware (including our enhanced sensor platform). The following diagram that shows the interaction between these components:

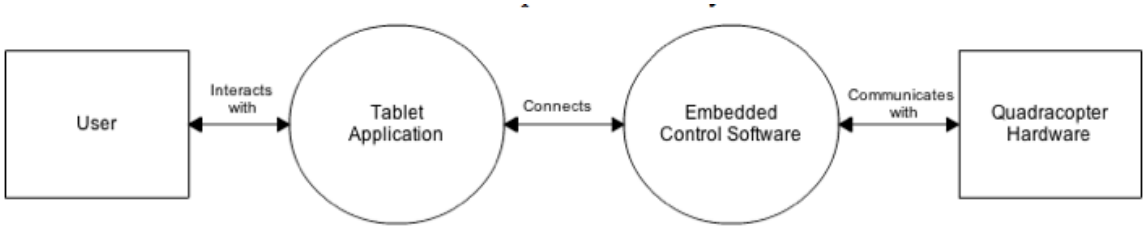

Figure 1: System Interaction Diagram

The system we have proposed is built from off-the-shelf components, in order to allow us to avoid spending time and energy recreating existing technology, such as the quadcopter and ultrasonic sensors. We are building our collision avoidance software on top of a Parrot AR.Drone 2.0 quadcopter, which already ships with control software that maintains all the onboard systems, and allows for easy and stable flight. The quadcopter employs a built-in, lightweight Linux-based operating system, which it uses to run the autopilot software as well as a WiFi wireless networking interface. Off the shelf, the autopilot software is responsible for responding to operator commands (such as “fly left, 20% speed”) by adjusting the rotors’ velocity. The onboard autopilot also maintains altitude using an onboard altimeter, and can “hold position” using computer vision analysis of the image from a downward-facing camera.

To support our collision avoidance system, we have purchased several range sensors from a Calgary-area vendor called Phidgets. These sensors are ultrasonic, and provide us with distance readings between 154 mm and 6.5 m. These sensors work out of the box, and require no user setup - we simply connect the “+5V”, “Ground” and “Data” lines to a Phidgets USB sensor controller, and read the reported distances from a simple C api.

Lastly, we will be using an Android Nexus 7 tablet with an open source tablet application that we will create to fit our needs. This tablet was selected as it is a stable, basic Android tablet, and is representative of the wide variety of Android tablets that could potentially host our control software in the future.

Modified Open-Source Android Piloting Application

The android application for this project will be built off of an existing open source app called "FreeFlight2", created by Parrot. This application is a fully functional app that can be used to fly an unmodified AR Drone. The application will be modified and only the essential features needed for flight will be kept. Along with these modifications we plan to add several features that are specific to our modified AR Drone. The existing application is quite complicated in the way that it has been built but this has been for a good reason, since the developers at Parrot did not want to write separate code for every deployment target. The android application specifically uses the AR Drone's flight API, which has been coded in C in order to be cross platform compatible. In order to compile this code for android, the NDK (Native Development Kit) provided by Google must be used. The GUI orientated part of the application however is written in Java and is built off of the android SDK. As it would not be wise to mess with the essential flight code of the drone which has likely been tested by parrot to be free of major bugs we will implement the core of our new features in java.

We intend to add three features to the software:

1. Upon connecting to drone, the user will be able to remotely launch the quadcopter's onboard autonomous collision avoidance software.

2. Addition of sound to alert the operator when a collision has been avoided, or the onboard autonomous collision avoidance software is proactively limiting the quadcopter's speed

3. A video overlay with invisible bars around the edge of the screen. These invisible bars would change color to non-intrusively warn the operator when the autonomous collision avoidance software is proactively limiting the quadcopter's speed, or the quadcopter is in danger of a collision.

Onboard Collision Avoidance Autopilot Software

The onboard collision avoidance software is responsible for preventing the operator from crashing the vehicle. This will be accomplished by intercepting operator commands, modifying them based on the ultrasonic sensor data, and then retransmitting them to the OEM autopilot software that ships with the AR.Drone.

The autopilot software is written in C++ and cross-compiled to ARM from a standardized Debian virtual machine image. This allows us to work on the code concurrently, using modern development tools, and write scripts for tasks such as compiling and deploying the software image.

Command Interception and Modification

The Parrot AR.Drone is operated by sending a series of flight commands over WiFi, via UDP packets. Our autopilot software intercepts these via a Linux iptables firewall rule, which redirects these commands to a port monitored by our software. Once the commands are received, we decode them, modify them, and then re-transmit them to the onboard autopilot software. The iptables rule is installed only on the wireless network interface, not the local loopback interface, preventing the commands from getting stuck in an infinite loop.

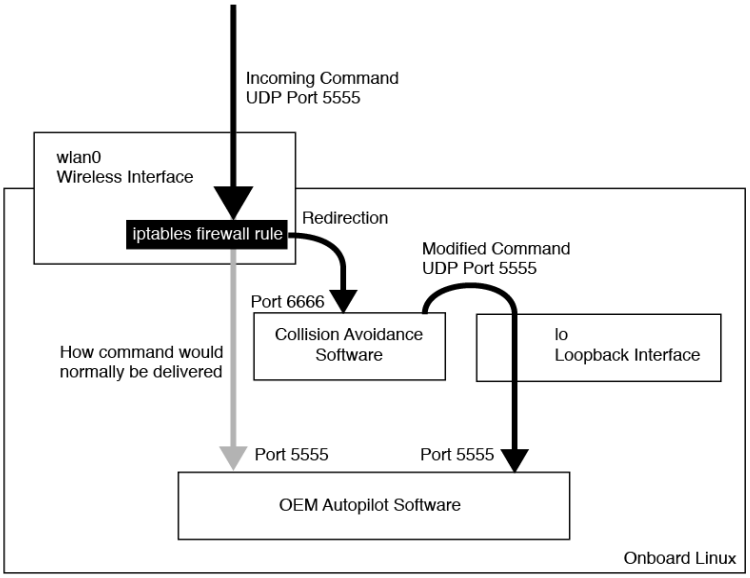

Figure 2: Command Path Diagram

The AR.Drone communication protocol is relatively straightforward. For flight commands, a UDP packet is sent with the following format:

```
AT*PCMD=%d,%d,%d,%d,%d,%d<CR>
```

The six values are as follows:

1. The packet sequence number
2. The current flight mode (hover, fly, fly with magnetic control, etc)
3. Drone left/right tilt
4. Drone front/back tilt
5. Drone vertical speed
6. Drone angular speed (spin)

Our collision avoidance software will intercept these commands, and modify flags 3 and 4 (left/right tilt and front/back tilt) based on the collision avoidance algorithm described in the next section. Once new values are computed, the command packet is recreated, and forwarded to the OEM autopilot. As far as the OEM autopilot is concerned, the operator original operator just

happened to avoid the wall. This highly decoupled design allows us to ensure we don't need to modify the existing autopilot's flight, stability, ground tracking, or safety algorithms.

Collision Avoidance Algorithm

The collision avoidance algorithm used is a simple dynamic limiting function, implemented against smoothed sensor data. As the quadcopter approaches an obstacle, its maximum forward speed is limited based on its distance to the object. Once the quadcopter gets within a predetermined "danger" radius, the maximum forward speed is reduced below 0; i.e. the maximum forward speed is actually backwards. This causes the quadcopter to back away from the obstacle, preventing a collision.

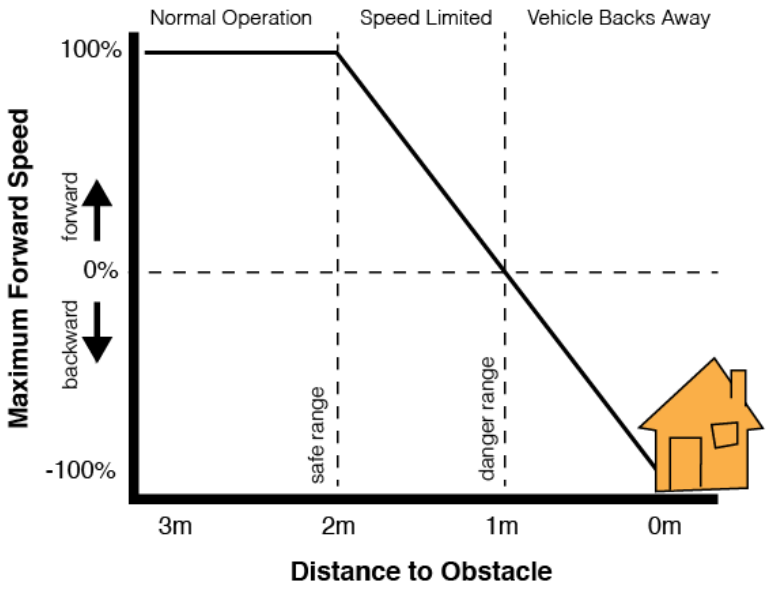

Figure 3: Quadcopter Speed versus Obstacle Distance for One Obstacle

The maximum reverse velocity is limited in the same way, based on data from the rear sensor. In the case where the maximum velocities overlap, such as if the quadcopter is constrained between two walls, the average of the two maximums is used. This allows the vehicle to automatically center itself in a crowded environment, without needing operator input.

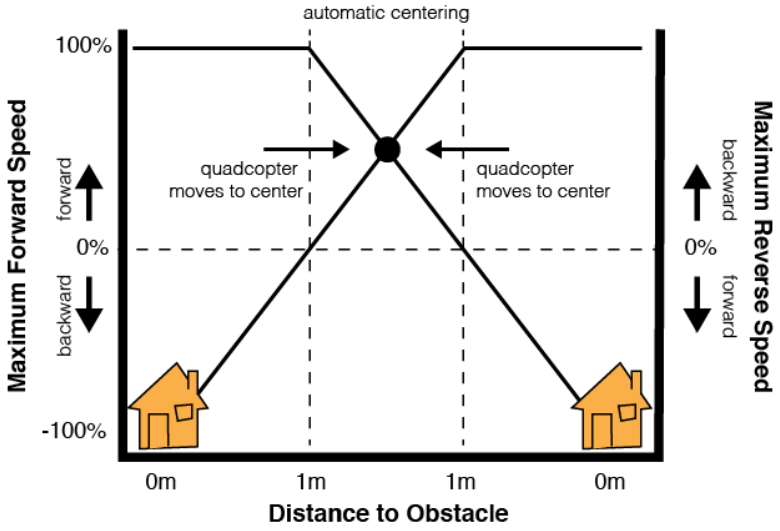

Figure 4: Quadcopter Speed versus Obstacle Distance for Multiple Obstacles

Once the front/back tilt is calculated, the same algorithm is used to calculate the left/right tilt. This allows the vehicle to avoid collisions along both of its lateral degrees of freedom. The OEM autopilot software already maintains altitude above the ground, and ceilings or other obstacles above the vehicle have been determined to be outside the scope of this project.

Code Structure and Design

The code is structured using a dependency injection (DI) technique, allowing us to separate the concerns of various modules, without coupling them more than is necessary. Any unrelated modules communicate through C++ interfaces (classes consisting of only pure virtual methods), allowing us to limit their coupling, and work on different modules independently. For example, the sensor data is only accessible through the I_SensorReader interface, which consists of a single “reading()” accessor function.

A complete diagram of the classes is as follows:

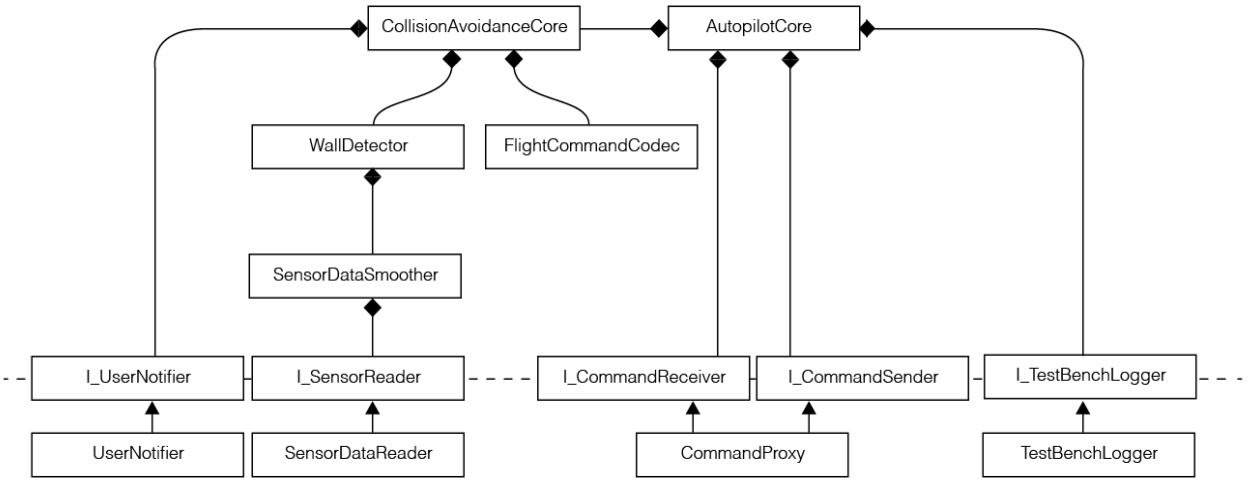

Figure 5: Class Diagram

Dataflow through the software mirrors the class design. Commands are read in from the UDP socket, modified based on the smoothed sensor data, and then written back out to another UDP socket, as in the following diagram:

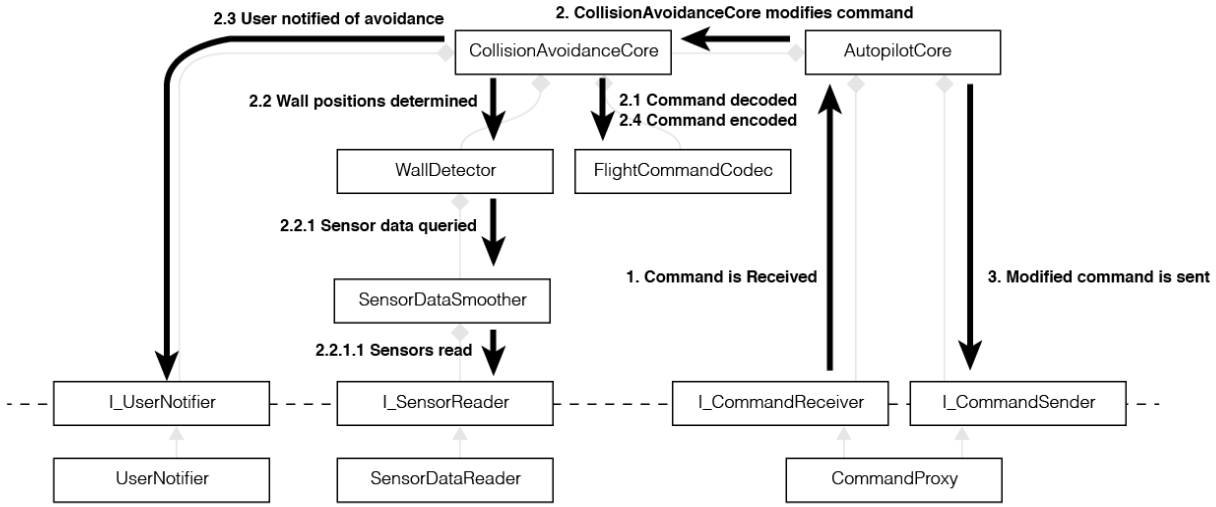

Figure 6: Data Flow Diagram

Hardware Design

We will use the original AR Drone hull and attach ultrasonic sensors to its outer edges. The sensors will communicate to the supplied Phidget Interface kit and communicate distance information to the tablet. The tablet will interpret this information and test for potential collisions.

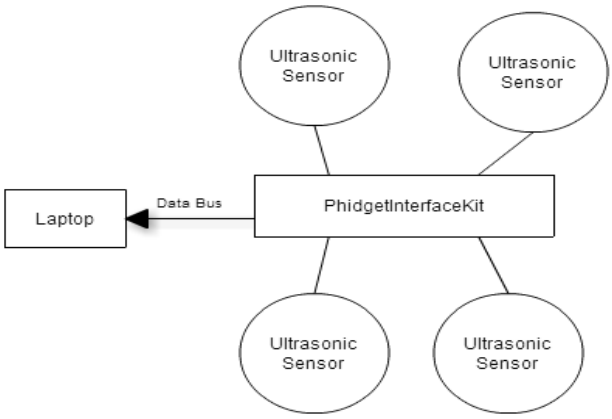

Figure 7: Sensor Connection Layout

Our initial hardware configuration had an ultrasonic sensor on each of the 4 edges of the frame. After initial testing we found that this was insufficient, giving us a blind spot at corners of about 10 degrees, as shown in the following diagram.

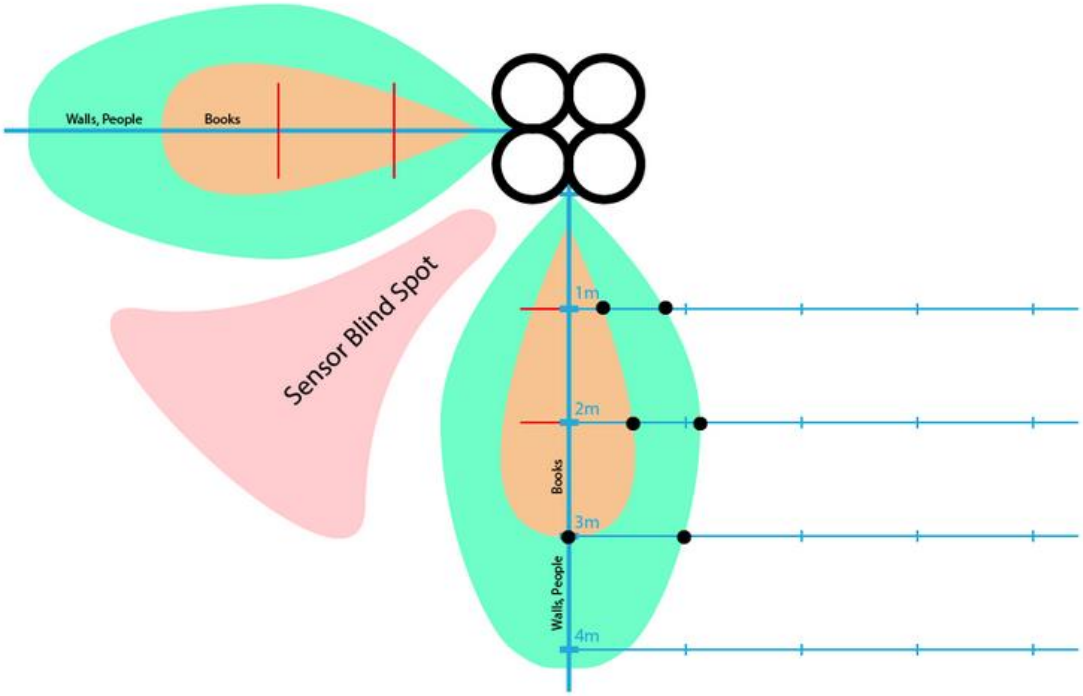

Figure 8: Former Layout with Ultrasonic Blindspots

To resolve this issue, we are adding additional ultrasonic sensors to cover the blind spot, allowing us to satisfactorily detect threats from all directions. Note that since the sensors have a conical zone, we will also detect obstacles from above and below. The new sensor arrangement is shown in the following diagram.

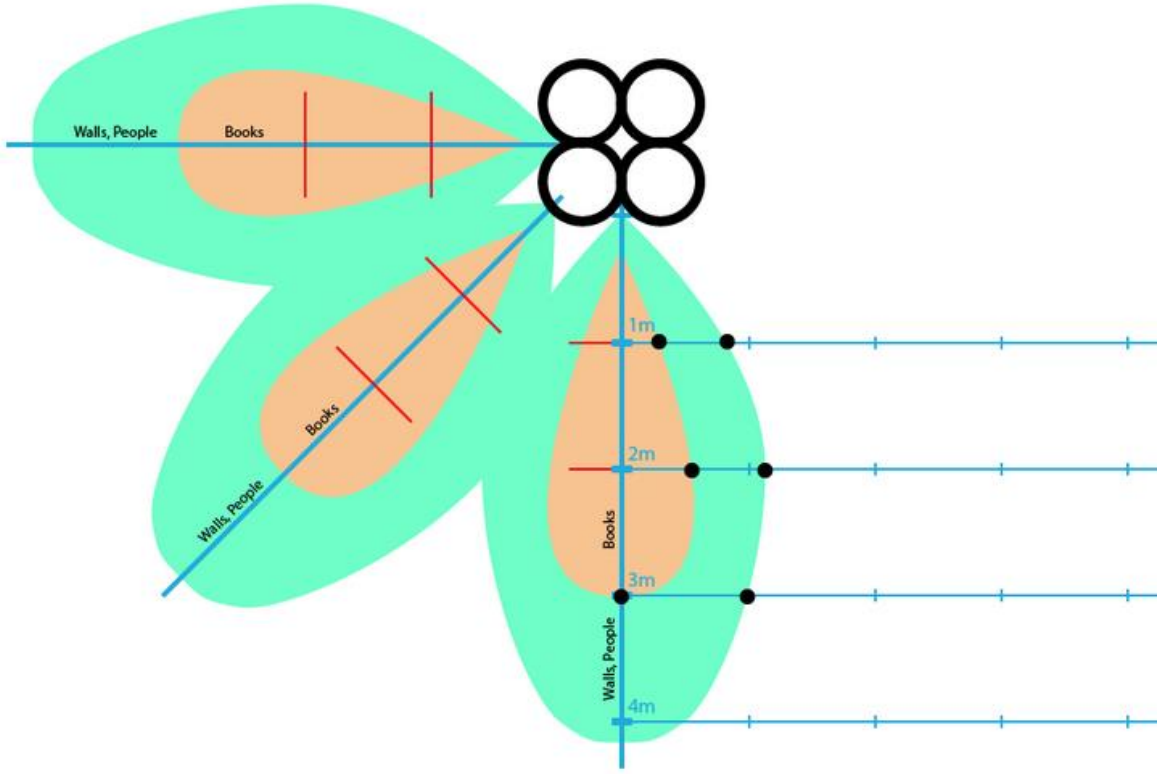

Figure 9: Updated Layout With Full Ultrasonic Coverage

Testing Plan

Android Application

The android application will be tested with the following methods:

1. Drone will be started along with the android application, using feature addition 1 we will load the collision avoidance system onto the drone. The drone will then be flown towards an object using the modified android application.
2. The drone while flying will be flown at its minimum speed towards an object. As the drone enters the specified collision warning zone the red bars laid out around the screen will fade in red as the drone gets closer to the object.
3. The drone while flying will be flown at its minimum speed towards an object. As the drone enters the collision zones tones will be listened for at increasing volume as we head towards the object.

Onboard Autopilot Software

The autopilot software was built using an Automated Acceptance Test Driven Development (AATDD) approach originally developed by our sponsor, Lockeed Martin Canada CDL Systems. This approach is similar to TDD, but on a higher level. Before any functionality is implemented, an automated acceptance test is written that tests this functionality, by only stubbing out external parts of the software. For our testing strategy, the following software components are replaced with mock objects for testing:

- **CommandProxy**, which wraps the UDP send/receive sockets used to communicate with the control tablet and the OEM autopilot
- **SensorReader**, which wraps the libphidget library used to communicate with our sensors
- **TestBenchLogger**, which wraps the console used to communicate debugging data to the operator

A helper class called “FakeArDrone” constructs these three mock objects, as well as the entire collision avoidance core system. This allows unit tests to easily create and destroy the C++ objects, without having to create them individually.

Once the FakeArDrone has been created, a human-readable acceptance test can be created. For example, a very basic acceptance test that ensures the AR.Drone utilizes data from the front-left sensor might read as follows:

```
TEST(UsesClosestForwardSensorReadingForCollisionAvoidance)
{
   // Arrange
   FakeArDrone::setup();
```

```
    placeWallAtFrontLeftOfVehicle(DANGER_DISTANCE);

    // Act
    sendCommand(flyForward());

    // Assert
    CHECK_EQUAL(proxiedCommand(), stop());
}
```

This test consists of three parts:

- **Arrange**, in which we ask the singleton FakeArDrone class to setup a new instance, and then place a wall in range of the front-left sensor.
- **Act**, in which we order the drone to fly forward
- **Assert**, in which we ensure that the proxied command, which is the one sent to the OEM autopilot, is actually a “stop” command due to the obstacle.

These tests are written using a number of “helper” functions designed to make them easily readable. This allows our customer to review our test coverage, and understand what we are testing, without having to understand our entire system architecture. These functions are generally short and concise; for example, placeWallAtFrontOfVehicle(double distance) could read as follows:

```
void MessageHelpers::placeWallAtFrontLeftOfVehicle(float distance)
{
      arDrone().sensors().readings_[ForwardLeftSensor] = distance;
}
```

When the collision avoidance core asks the mocked sensor reader for the ForwardLeftSensor’s reading, it will return the value in the ‘readings_’ map, which in this case has been set to ‘distance’.

For the core autopilot system, we anticipate having the following automated acceptance tests:

TestSaysHelloToTestBench.cxx

```
TEST(SaysHelloToTestBench)
```

TestReportsSensorData.cxx

```
TEST(InitializesSensorsOnStartup)
TEST(ReportsSensorReadingsToTestBench)
TEST(ReportsSensorReadingsToAndroidTablet)
```

TestProxiesCommands.cxx

```
TEST(InitializesCommandProxyOnStartup)
```

```
TEST(ProxiesAllUnknownCommandsWithoutModification)
TEST(ProxiesMultipleCommandsPerApplicationCycle)
```

TestInterpretsCommandsCorrectly.cxx

```
TEST(CurrentPlatformUsesExpectedFloatingPointMemoryRepresentation)
TEST(InterpretsArDroneSdksIntFormattedFloatingPointNumbersCorrectly)
```

TestAvoidsWalls.cxx

```
TEST(DoesntCrashIntoWallInFrontOfCopter)
TEST(DoesntCrashIntoAnyWalls)
TEST(CanStillFlyAwayFromWalls)
TEST(FlysAwayAutonomouslyWhenWithinDangerDistanceOfWall)
TEST(DangerRangeWorksForRearSensorsToo)
TEST(AutoCentersWhenFlyingBetweenNarrowWalls)
TEST(NotifiesTabletWhenCollisionsHaveBeenAvoided)
```

TestSupportsDiagonalSensors.cxx

```
TEST(ReportsForwardAngledSensorDataToTestBench)
TEST(UsesClosestForwardSensorReadingForCollisionAvoidance)
TEST(UsesClosestLeftSensorReadingForCollisionAvoidance)
TEST(UsesClosestRightSensorReadingForCollisionAvoidance)
```

TestExitsHoverModeWhenInDanger.cxx

```
TEST(LeavesHoverModeWhenWithinDangerRadiusOfAWall)
TEST(DoesntAffectHoverModeWhenEntirelySafe)
TEST(DoesntAffectHoverModeWhenWithinSafetyZone)
TEST(IgnoresUnintentionalCommandsWhileOverridingHoverMode)
```

TestSmoothsSensorData.cxx

```
TEST(InstantaniousSensorReadingsDontScrewThingsUp)
```

As we develop and enhance our autopilot software, we may add tests to this list as necessary. We believe this set of tests covers our functional requirements, but it is easy to add more as needed.

Hardware Test Plan

We constructed a series of tests to determine the accuracy and sensitivity of the ultrasonic sensors attached to the quadcopter. The quadcopter was connected to the laptop through Wifi and was constantly outputting the distance data from each of the sensors equipped to the quadcopter. From there we used various sized objects and a measuring tape to test the distance that the sensors reported. The objects used include a small sized box, a standard sized binder, a larger piece of cardboard, and a human body. The following types of tests were performed.

1 This test was performed to determine if the distance data returned by the sensors was relatively accurate with real life measurements. For this test we used objects of various size and placed them in front of the sensor. The objects were then moved farther away from the sensor and the reported distance was compared to the actual distance.

2 This test was to determine at what distance the sensors stopped returning accurate data. This test involved moving objects of different sizes away from the sensor until the reported distance remained constant. This data will give us the approximate distance that the quadcopter will be able to detect a wall from.

3 This test was to determine the blind spots of the sensors. Our quadcopter has sensors mounted on the front, back, and both sides. We wanted to determine where the blind spots were and whether it would be justified to add more sensors. This test involved bringing in objects of various sizes towards the sensor from the side, and then determining at what distance the sensor began to detect the object.

Budget

A) Budget outline

	Budget request
Materials and supplies	$5000
Software	$0
Small equipment	$0
Travel	$0
Books and journals	$0
Subscription to resources	$0
Consultant fee	$0
Others; specify:	
Others; specify:	
Others; specify:	
Total	$5000

B) *Budget justification*:

We need this budget in order to purchase several quadcopters, various sensors with control boards, and an Android Tablet for a control station. The budget also includes money in order to fix the quadcopters if they crash or become damaged during testing.

C) Source of funding:
Lockheed Martin Canada CDL Systems

Health, Safety and Environmental Issues

Health, safety and the environment is always a big issue in engineering projects. Our project is primarily a software project but since we do have hardware that can fly through the air we have quite a few issues that need to be recognized so that they may be prevented.

Potential Hazards

1. The Quadcopter could potentially hit a bystander or an obstacle if the control software incorrectly fly's the quadcopter the wrong direction when attempting to avoid terrain. This could result in minor property damage or injury. We do not foresee any significant risk of death or disfigurement in the event of a personnel or bystander collision.
2. The Quadcopter while flying could be attacked by large birds of prey in the area. The effect of such a collision has the potential to harm said large bird of prey along with the quadcopter falling from the sky and possibly hitting a bystander or damaging university property.
3. Repetitive stress or carpal tunnel syndrome could occur in one or many of our team members due to excessive typing while trying to finish the project or due to excessive flying using the tablet.
4. Security risks are present due to the 2 cameras on board the UAV. As such, the aircraft should not be flown over private property without permission. All personnel and bystanders in the vicinity should be be made aware if the video will be recorded and used for purposes other than the navigation of the aircraft.

Standards and Regulations

Section 101.01 of the Canadian Aviation Regulations (CARs) states, "Unmanned Air Vehicle" means a power driven aircraft, other than a model aircraft, that is operated without a flight crewmember on board. Additionally, it must also weigh 35 kg or less. By this definition, we consider our product to be a UAV.

In Canada, if the UAV will be used for profit, or owned by a company, a Special Flight Operations Certificate (SFOC) is required. This license is free to obtain with a simple application process. The application requires that the operator proves to Transport Canada that they will not be putting the public in danger nor will they be disrupting air traffic.

Additionally, Transport Canada advises that a 100-foot horizontal buffer be kept between the aircraft and a crowd of people. An emergency landing location must be available, and all possible effort must be made to keep the craft from losing control while in the air.

Work Plan

Tasks	Deadline
The low level design is approved and finalized	January 25th, 2013
The specifications are approved and finalized	January 25th, 2013
The test plan is approved and finalized	January 25th, 2013
List of required parts and materials are finalized	January 25th, 2013
All parts and materials are available by	January 25th, 2013
First version of the project is implemented	January 28th, 2013
First iteration of testing is completed	February 11th, 2013
Further improvements of design and/or test plan	February 30th, 2013
Final version of the project is implemented	Mar 15th, 2013
Testing of the final version of the project is completed	Mar 20th, 2013
Final presentation is ready	Mar 27th, 2013
Report #2 is ready	Mar 24th, 2013
The final version of the poster is printed off	Apr 5th, 2013
Capstone design fair (*Tentative*)	Apr 10th, 2013

References

1 “Unmanned Aircraft Regulations”, Transport Canada. http://www.tc.gc.ca/eng/civilaviation/standards/general-recavi-brochures-uav-2270.htm

2 “LV Maxbotix EZ1 Specifications Sheet”, Phidgets. http://www.phidgets.com/documentation/Phidgets/1128_0_EZ1-Datasheet.pdf

3 “AR.Drone Specifications Sheet”, Parrot AR.Drone. http://ardrone.parrot.com/parrot-ar-drone/en/technologies

Glossary

CDL Systems - The former company name of our sponsor. The CDL does not stand for anything.
GUI - Graphical user interface.
OEM - Original Equipment Manufacturer
UAV - Unmanned aerial vehicle.
USB - Universal serial bus.
Quadcopter - Also known as a quadrotor; a multicopter with 4 rotors.
SDK - Software development kit.
NDK - Native development kit.