**AggiE-Challenge**

**Autonomous UAV**

**Executive Summary**

**May 13, 2014**

|  |  |  |
| --- | --- | --- |
| Bradford Stricklin | Jui Yen Chua | Allison Badgett |
| Dayton Savage | Spencer Davis | Dennis Moore |
| Michael Kowpak | Adway Dogra | Nathan Kocmoud |
|  | Fuhua Song |  |

**Abstract**

The objective of this project is to design and build an unmanned aerial vehicle that is capable of autonomously navigating the interior of a commercial building and collecting accurate occupancy information. To solve this problem, the group will use multiple quad copters working at two stages. The first stage will use one UAV to navigate the building and generate a 2 dimensional map of the layout. The second stage will use the remaining UAVs to follow the map and count the number of occupants. The group was split into three teams to focus on the three main aspects of the project: navigation for discovering and mapping the building layout, enumeration for getting occupancy counts and information, and hardware for developing the UAV with appropriate flight control and obstacle avoidance features.

The red text in this report represents the work completed during the Spring 2015 semester.

**Navigation**

The job of the navigation team was to solve all of the problems related to the movement of the UAV, mainly the issue of how to explore and map an unknown environment. The first step in our design was to determine valid solutions to the exploration and mapping problems by doing copious amounts of research into different possible methods. The research we did was focused around using a laser range finder for position tracking, as we already had access to an expensive laser range finder. In order to save time, we decided to use the Hexacopter built by last years Aggie E challenge team; this allowed us to worry more about the software side of the project and gave us more time to come up with better solutions to the navigation problems.

After a few weeks of research, we decided on a simple wall following technique to handle the exploration. The 2D mapping would be handled using Python's matplotlib library. Before we could start testing our solutions, however, we needed to decide on the specific hardware we were going to use. Together with the hardware team, we decided to use a Raspberry Pi as the main computer that would link together the different hardware systems while also having the autonomous control system on it. Along with the Pi, we decided to use an Arduino Mega for data acquisition and filtering. To avoid having to develop our own low level flight controller to calculate attitude and control the motors, we also purchased the APM 2.6 flight controller. The Arduino Mega is currently connected via USB to the Raspberry Pi, and the APM 2.6 will be connected to the Pi via a serial connection at the beginning of next semester. The following sections describe the work done on autonomous exploration and mapping in further detail.

Exploration/Autonomous Control:

Our preliminary research led us to believe that a wall following technique would be the best way to explore an unknown room, and it had the added benefit of being relatively simple. To test the wall following method, we created a simulation in LUA of a UAV exploring different shaped rooms, and it was a great success. The worry with the wall following method is that we might miss objects in the center of the room, but with the assumption that each room will only be 40ft x 40ft, this should not be an issue since our laser scans 20ft. This has been tested in simulation, but we plan to test it on the UAV at the start of next semester. For the autonomous control system, we have decided to use a PID controller to move us along the wall and decrease our throttle as we get closer to an obstacle (wall, corner, pillar, etc.). We will use the URG-04LX to do position tracking, and the APM flight controller to give us the attitude(roll/pitch/yaw) of the UAV. We plan to limit the range of the UAV's movement to make the autonomous control simpler, i.e. the UAV will only be able to move forwards, backwards, left, and right. Also, we plan to add an altitude controller to the APM that will hold the UAV at a certain height. This will be the bulk of our work next semester.

Mapping:

Our first idea for handling the mapping problem was to use Python's matplotlib library to plot the distance data received from the URG-04LX laser range finder. The URG-04LX scans a maximum of 5 meters over a range of 240 degrees, so XY coordinates can be obtained from the distance and angle by doing some simple geometry. In our first mapping attempt, the URG-04LX was connected to the Raspberry Pi via USB, and the data was read using a C library from Hokuyo. This data was then saved to a file and plotted in excel. The final goal of this method was to write Python code to create a 2D map, and to use shared memory to access the sensor data. The problem with this method, though, is that additional code needed to be written to referentially localize the UAV within the room and overlay the maps generated by each laser scan. In order to solve these problems, we decided that using a ROS (Robot Operating System) library called Hector Mapping would be best. Hector Mapping deals with these issues by utilizing SLAM(Simultaneous Localization And Mapping). Since ROS will not run on the Pi, we are planning to run ROS on a host computer and send the URG-04LX data from the Pi to the host computer wirelessly. This will be accomplished next semester.

This semester the navigation team worked on completing autonomous flight control and two dimensional Simultaneous Localization and Mapping (SLAM).

Autonomous Control:

The ROS package "roscopter" was utilized to receive IMU data from the APM, but it was unable to be used for overriding the RC commands as previously thought. In order to override the RC commands, PWM signals were sent from the Raspberry Pi's GPIOs straight to the APM's RC inputs. These PWM signals were multiplexed with PWM signals from the RC receiver to allow for manual override. A ROS node was then written to accept keyboard input (W, A, S, D) from a host computer and generate different movement commands based on this input. Thus wireless communication and control was achieved between the APM and a host computer.

Simultaneous Localization and Mapping:

The ROS packages "hokuyo node" and "hector mapping" were used for the SLAM. The "hokuyo node" package received data from the laser range finder and published it on the "/scan" topic. This topic was read in by RVIZ and the hector mapping node to create a constantly updated map based on the laser range finder's readings. A sonar sensor was also interfaced with the Raspberry Pi in order to detect windows, as laser beams will pass through glass.

Furthermore for mapping, we wanted a way to detect obstacles that the laser range finder might not be able to pick up. We decided to utilize an ultrasonic sensor for this. The ultrasonic works by measuring the time it takes for a pulse to be sent and received back. Then it turns this time into distance. The outcome is data displayed in feet and the median value of three numbers are chosen as the distance to avoid making any errors. Furthermore, there's a distance limit to how close the sensors is to the object it's detecting. The object can't be too close or the data output is incorrect. However, when utilized on the UAV, it should serve as an adequate secondary to the laser range finder and would have no problem detecting windows. The next step however is to integrate it into ROS.

**Enumeration**

The objective of the enumeration team is to obtain a 100% accurate occupancy count as well as occupant locations in case of an emergency. We will be using a mounted video camera to detect people inside the building, and this video feed will then be sent via antenna to be analysed on an off-board computer. Because of the large scale of the necessary enumeration program, we determined that using an on-board processor would not be practical. The program will still work autonomously to determine the number of occupants as well as their time-stamped locations and present these at the end of the flight. Runtime user input will not be necessary. As of this time, we still need to purchase the antenna for the camera and the antenna-USB connector for the computer.

Within the enumeration portion of this project, there are two major subtasks: counting/tracking and implementing a search algorithm.

Counting/Tracking

The major challenge within this portion of the project is detecting people in several different positions and states of motion. We will be using Matlab's Computer Vision Toolbox to detect occupants using multiple built-in functions designed for this purpose. We will use the PeopleDetector constructor to create an object that identifies upright people. In addition to this, we will use the CascadeObjectDetector constructor to create an object that can identify facial features and the upper bodies of seated people or those who do not fit the model detected by the PeopleDetector. In many cases, multiple features will be detected on the same person. In order to avoid counting one person multiple times, we will use the Computer Vision function bboxOverlapRatio in order to determine which features overlap and then eliminate redundant counts.

Similarly to the procedure stated above, we will be using MATLAB’s Computer Vision Toolbox to track the people that were previously identified in order to avoid redundant counts. From there, numerous other features – such as continuously adaptive mean shift (CAMShift) and Kanade-Lucas-Tomasi (KLT) – that are readily available can be used to track single or multiple objects in a video stream. For instance, a background subtraction algorithm that is based on the Gaussian mixture model is used when tracking objects. The accuracy of tracking is further enhanced through reducing the noise and grouping the desired pixels.

To implement the counting algorithm, we decided to use the tracks created by the People Detection algorithm. However, after evaluating the corresponding results and inspecting the MATLAB code, we found out that the tracks created by MATLAB may or may not contain useful information. This implied that certain modifications had to be done. As a result, we implemented two counters that will count the number of bounding boxes created in each video frame: one of the counters contains the count of people in the previous frame, while the other one contains the count of people in the current frame. A comparison is then made between the two counters, and if the latter is greater in amount than the former, then the difference between the two counter values will be added to the global counter, which indicates the effective number of people detected so far.

A challenge to the strategy mentioned above is to increase the accuracy. Due to the fact that the algorithm is very simple and not very sophisticated, possible errors may occur. For instance, the result may be inaccurate when two people captured in the video appeared to be two for the first consecutive video frames, but then appeared to be aligned as one, and later separated into two individuals again. This may be something the algorithm is unable to determine. This problem can be tackled through changing the UAV's flying altitude and the camera's posing angle. Besides, MATLAB's algorithm is considered to be very sophisticated, since it is able to allow bounding boxes to overlap one another for a few consecutive video frames, hence the possibility for this error to occur is actually minimal.

Video Streaming:

A Raspberry Pi and Raspberry Pi camera module are used to stream video from the UAV. A Python script using the picamera library controls video stream time and resolution and sets up a TCP socket for streaming over a WiFi network. On the base computer, the command-line application VLC media player reads the streaming video into a file. After the flight, Matlab will retrieve this video file to be processed using techniques described in the previous section.

Search Algorithm:

Before the search algorithm is applied, the points from the 2D scanner are overlaid onto a grid of 4" x 4" cells to simplify the map. 4 inch cells are optimal because it is a standard size for mapping grids in robotics, it is big enough to simplify the search algorithm, and it is small enough to enable navigation through doors and basic object avoidance. This grid is then converted to an array of 1's and 0's and passed to the search algorithm with the starting point.

The algorithm that will search the building space has a great deal of importance, and a number of algorithm ideas were considered. Ultimately, the algorithm that worked well for the largest number of building layouts was chosen. The Zhang-Suen image decomposition process reduces an image to its skeletal frame. When applied to a building layout, the algorithm produces a pathway that will pass through each room and hallway. This path can then be traversed by the UAV on the enumeration sweep. For very complex layouts, key points can be picked out along the path and the A\* pathfinding algorithm can be used to navigate between them. Skeletal decomposition greatly simplifies selecting these “critical points.” The endpoints of lines are nearly always located in the center of rooms or alcoves. Junctions where pathways meet are also easy to detect. These key features will be connected to define the pathway the UAV will follow.

**Hardware**

Project Objectives: The hardware team’s mission is to ensure the UAV is fully functional, integrates the sensors from the navigation and enumeration teams, and is structurally stable.

The UAV began as a hexa-copter and was later changed to a quad-copter due to size constraints. A standard entryway is 36" and the hexa-copter size was too large to navigate with autonomous control. The new quad copter design provides a comfortable clearance through doorways allowing room for slight error in guidance while the original design would of required precise movements and detailed environment awareness. This design change came at the cost of the extra lift and stability the hexa-copter offers. However, for the current application, the quad-copter design is preferable.

Switching to the quad-copter design reduced the amount of weight the UAV could carry and reduced the surface area for attaching sensors. With this in mind layout will be organized for an even weight distribution. This aspect is still subject to change depending on where certain parts that have yet to be attached are mounted. Weight capacity will be recalculated by Michael Kowpak when all components are ready to be mounted to the UAV.

The hardware team has also ensured that the UAV can actually fly. The quad-copter was assembled using four of the arms and motors from the original hexa-copter and soldered all necessary connections from the motor, battery input, and flight system components. Using the ArduPilot flight controller and the Mission Planner software the motors and hardware attached up to this point are functional and working properly. The current assembly is a basic functioning quad-copter and consists of the ArduPilot 2.6 flight controller, a compass, GPS and radio communicator for the UAV. At this point in the project the UAV can be tested/controlled via a radio controller.

Next semester, the hardware team will use the Arduino program and ArduCopter software to allow for autonomous control. Once the quad-copter can fly using command inputs instead of a handheld remote control, the sensors from the enumeration and hardware team will be integrated into the design.

**Autonomous Flight:**

This semester, the hardware team worked on getting the Mission Planner software up and running. The biggest problem we faced last semester was not being able to set up the flight controller. This was primarily because we have a defective flight controller. This semester, however, we were able to run the Arducopter 3.2 firmware on the APM flight controller and carry out flight testing outdoors. We carried out manual fight early in the semester and autonomous flight later on. Initial autonomous flights were carried out using the barometric pressure sensor as the altitude hold sensor. Later, according to the navigation teams needs, we had to add a sonar sensor for altitude hold. The Maxbotix analog module was used for this. We verified that the altitude hold indeed worked because varying the distance between the module and the floor varied that throttle.

**Maxbotix Sonar:**

The altitude hold will be handled through the APM. The sonar sensor is enabled in Mission planner. The APM will receive an analog signal in the AUX-0 port from the Sonar Sensor. The sonar sensor is a MB1242 Module. The MB1242 has a max range of 7.65 meters and a resolution of 1 cm. The reason we need altitude hold is to keep the quad copter stable at a given height that will be pre determined. This constant height is a key component of the autonomous flight handle at the APM level. The sensor will feed a distance reading handled by the APM to manage thrust related to is height position from the ground. The sonar sensor is mounted under the center body of the quad copter and is pointing straight down normal to the ground surface.

**MUX failsafe selector switch / failsafe switch:**

The hardware tem also designed a multiplex failsafe switch for the UAV. Since this is an early prototype, crashes due to software bugs is highly likely. The switch controls the input going into the flight controller and switches it between the RC receiver or the raspberry pi. Connecting the Raspberry Pi on the input pins was a slight change from the proposed design last semester. We had initially planned to use ROScopter and MAVLINK to control the flight controller serially via USB. However, there were problems getting the ROScopter to control the UAV. This was why we switched to mimicking a PWM signal on the Pi and connecting it to the RC receiver input pins but multiplexing with the actual RC receiver. The default configuration of the switch is latched to the the RC receiver, which means that it can only be controller by the RC transmitter. The raspberry pi can, however, control the select switch and latch the control lines towards itself. In this case the Pi asserts full controls over the UAV. In case of a malfunction, the Raspberry Pi select line will go low and the control line will get latched back to the RC receiver for manual control.