Control of a Multirotor Swarm through Guided Autonomy

Maximillian Schommer
Eric Miller
Paul Nadan
Jamie Cho
Adam Selker
Nathan Estill
Franklin W. Olin College of Engineering

ABSTRACT

To accomplish Mission 8 of the International Aerial Robotics Competition, we have designed an aerial robotic swarm capable of both local autonomy and non-electronic control by a human operator. We use a series of 4 modified Bebop 2 drones, each with a front facing camera and MEMS sensor suite. Additional ultrasonic rangefinders have been added for obstacle avoidance and threat detection. Each drone communicates with a base router over WiFi and a single central computer handles autonomous control of the swarm. Drones navigate on a low level using an onboard flight controller, while receiving high level instructions from both voice control and gesture recognition interfaces. We here describe our system in detail, including design of the robotic system, control, perception, modeling, and user interface.

INTRODUCTION

Statement of the Problem

In order to solve Mission 8, we must create a highly integrated intelligent robotic system. First, each robot must be capable of avoiding obstacles and enemy robots and navigating the arena without the aid of GPS or nearby features useful for SLAM to function. Second, all robots must be capable of operating collaboratively to aid the person in retrieving an item from a set of four bins. They must be able to locate the bins, and then simultaneously image a broken code on the top of each bin and reassemble it in order to tell the person how to unlock the bins. The robots must also be capable of operating semi-autonomously and they must take high level commands from the person in the form of gestures or voice control, while no direct electrical control is allowed. Each drone must also be person-safe, meaning that a finger must not be able to touch any of the rotating blades of the robots. This all must be completed within 8 minutes, and without the human being tagged with enemy lasers too many times.

Conceptual Solution to Solve the Problem

We have a single computer managing the robot swarm, and running each robot's controller as a thread. There is a single main thread which manages all of the robots. The operator in the arena gives vocal commands to a microphone. These commands are parsed by the main thread, and then interpreted as high level instructions for each robot or a set of robots to perform tasks, such as moving directions, identifying bins, processing QR codes, etc. Each robot has locally

autonomous behaviors, such as running person/object recognition, tracking and position estimation, and obstacle avoidance. The full system architecture is depicted in Figure 1.

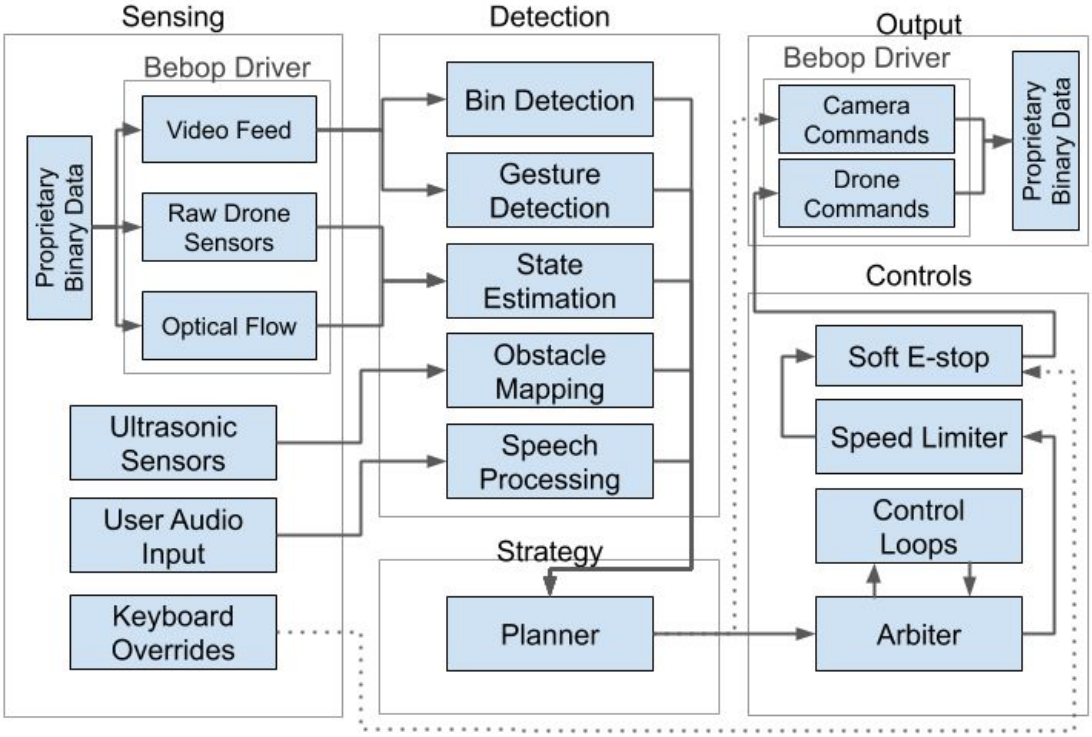

Figure 1. Overall system architecture.

Yearly Milestones

As this is the first year of IARC Mission 8, as well as our team's first year competing, we set out to accomplish all the tasks that are necessary to successfully complete the mission by retrieving the critical component. These included: computer-controlled flight of a drone swarm, obstacle avoidance, QR code recognition and stitching, non-electronic operator input, and required safety features. While these essential tasks have all been successfully completed, several other tasks were left for next year, including the healing laser and the autonomous delegation of tasks by the swarm. Both of these features, while useful, can be left out without rendering completion of the mission completely impossible.

AIR VEHICLE

Description of Configuration/Type

Since many of the hardware features we require were available on common systems, we decided to use a set of 4 Parrot Bebop 2 [1] aerial vehicles for our base platform. These are quad-rotor drones, which are capable of flying for 25 minutes unmodified. For each of our autonomous aerial vehicles, we chose to use a visual localization system, since LIDAR would not be useful without nearby features. There are two cameras on each robot, one facing directly down performing optical flow paired with a height sensor, and another facing forward. We use the optical flow camera and height sensor to obtain relative odometry. Each drone is also equipped with a custom EStop/Obstacle avoidance package which does not come standard. Because we do

not have the ability to connect additional sensors to the Bebop 2's flight computer, we instead send the additional sensory information over WiFi via ESP8266 modules. Figure 2 depicts the hardware onboard each of our Bebop 2 drones.

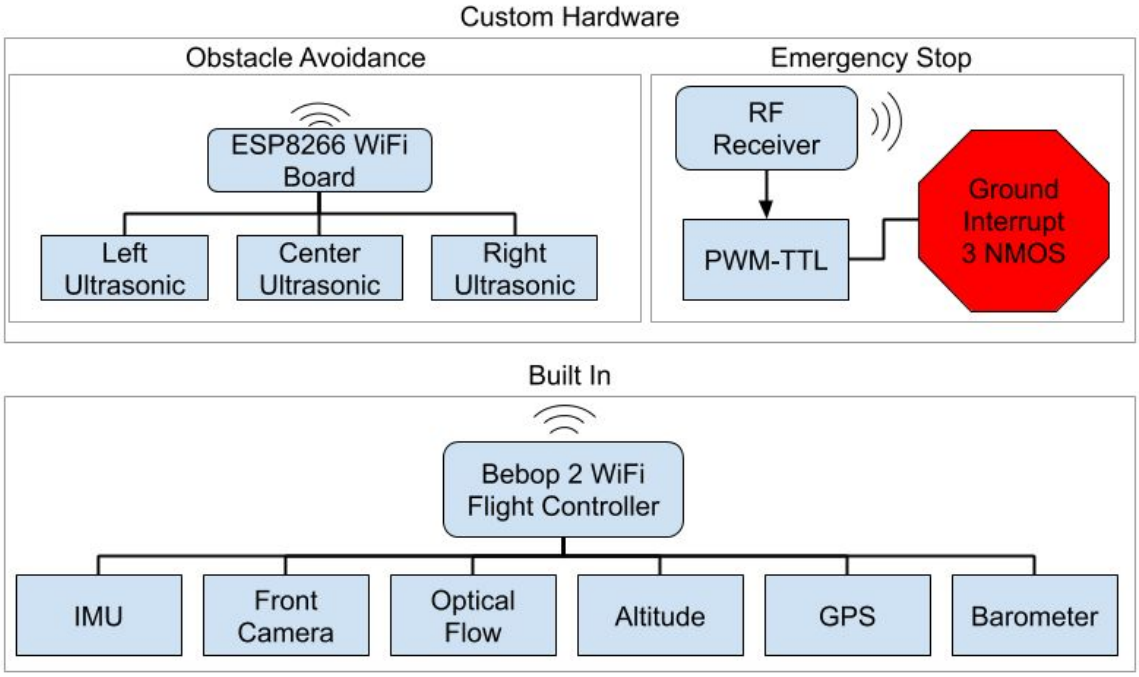

Figure 2. Sensor and sensory output configuration of modified Bebop 2

Flight Control System

Navigation/State Estimation System

The onboard visual odometry built into our drone, the Parrot Bebop 2, is fairly accurate in the local domain. In order to improve the robustness and precision of state estimation, we sought to employ additional optimization based on matching observations across different drones at different times. The system is currently unstable, but we expect to have a more robust implementation by the time of the competition.

Attitude/Position Control System

Our flight control system is built as an abstraction layer we refer to as the "arbiter" which is responsible for managing the division of labor between the underlying platform and control algorithms we write ourselves. For example, on the Parrot Bebop 2-based platform we use, the platform provides builtin control capabilities for low-level attitude control, but also higher level hovering capability and even scripted routines for safe takeoff and landing. Where available, we want to use these higher level commands because they execute with high performance on the drone, and have been highly optimized by Parrot.

On the other hand, the 2D simulator platform we developed (see Modeling and Simulation section) does not have built in takeoff and landing scripts, but does have the low-level ability to directly command vehicle velocities. We want to maximize the performance of our code on both

platforms, as well as any future drones we wish to work with that may offer a different set of capabilities.

To support this abstraction, the arbiter (of which one copy is executing per drone) accepts commands from high-level code over ROS topics in a variety of formats. The currently implemented set of command formats are shown in Table 1.

TABLE 1. HIGH LEVEL COMMAND FORMATS ACCEPTED BY THE ARBITER AS OF THE 2019 IARC COMPETITION

Input ROS topic	*Command type*	*Control loops*
/{name}/high_level/cmd_takeoff	[no data]	[none]
/{name}/high_level/cmd_land		
/{name}/high_level/cmd_vel	3D vector + yaw velocity	Velocity scaling
/{name}/high_level/cmd_vel_alt	2D vector + yaw velocity + altitude (meters)	Velocity scaling + Proportional vertical control
/{name}/high_level/cmd_pos	3D Position target (incl. coordinate frame)	PID lateral control + proportional vertical control
/{name}/high_level/cmd_rel_pos	3D position relative to current location	PID lateral control + proportional vertical control
/{name}/high_level/cmd_cam_pos	3D position target + 3D point to face towards	PID lateral control + proportional vertical control + proportional yaw control

For many of the command formats, the task of "transforming" the high-level commands into commands that can be understood by the underlying platform requires executing several control loops. To accomplish this, the arbiter uses a stack of composable Transformers that implement control loops, including separate PID loops controlling lateral position, altitude, and heading.

During normal operation, the arbiter (Figure 3) is the last piece of custom code to modify commands before they are sent to the drone's drivers, so we also use it as a place to apply soft velocity limits to reduce the likelihood that bugs elsewhere in the code inadvertently cause extreme behavior. This limit is implemented in the "pseudo-velocity" space exposed by the Bebop drone. We separately clamp the vertical commanded velocity and norm of lateral velocity, by if necessary dividing the commanded velocity vector by the norm of that vector, so the original direction of the command is preserved and only the norm is modified.

Flight Termination System

There are three ways to terminate flight, landing, soft EStop and hard EStop. Landing is gentlest, and begins the landing routine of the drone. This is initiated by the operator with a voice command to land, or by the computer operator who can command the drones to land directly. Next is soft EStop, which is a similar command through the same interface, but instead of initiating a landing sequence, it cuts power to the motors through the WiFi channel of the Bebop 2. Finally, as a method free from software, we interrupted the main ground line from the battery of the Bebop 2 and placed 3 parallel power NMOS in series with the battery. A radio transmitter sends a kill signal to the drone RF receiver. The PWM signal is converted to TTL via the method

prescribed in the IARC suggested kill switch design. Each drone has its own transmitter to terminate flight with this method.

Figure 3. ROS Node map of a drone.

MISSION PACKAGE

Perception System

Target Identification and Behavior

The four drones must image a QR codes on the top of four black bins simultaneously. In order to accomplish this with as little human intervention as possible, we proceed in three steps. First, a voice command is given by the operator to locate and fly to bins. Next, each drone runs a bin detection algorithm to find the locations of the bins and fly to them. Finally, once each drone is positioned above a bin, simultaneous imaging, QR stitching, and decoding occurs when another voice command is received.

In order to identify the bins, a series of tools have been created to generate training data to train a Haar Cascade classifier. First, we recorded a series of realistic training videos of the bins. We then created automated labeling tools in order to generate bounding boxes for the bins. This used a variety of algorithms, depending on the scenario, including GrabCut [2], adaptive thresholding, color segmentation, and edge detection. Once enough videos were labeled, we trained a Haar cascade classifier on positive images of the bin extracted from the videos taken (Figure 4). This included views from every angle of the bin, and the bin on different surfaces and environments. When properly tuned to reject outliers, the classifier performed sufficiently well for our purposes, and was used to capture initial instances of bins. Once an instance of a bin is captured, we can continue tracking it using a Distractor-aware Siamese Network (DaSiam) [3].

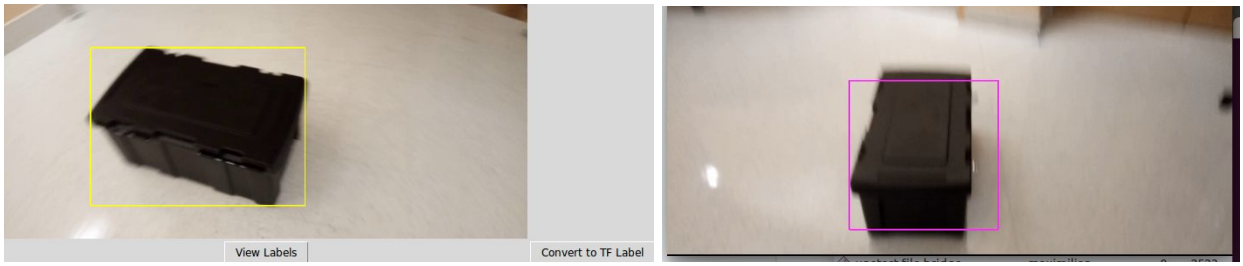

Figure 4. Left: Labeling tool. Right: Trained Haar cascade classifier

Once a drone is aligned over every bin, the QR stitching algorithm is run on the set of images they take simultaneously. First, the algorithm uses a threshold to get all of the pixels that are white on the tablet screen. Then it uses the resulting contour to get a box around the QR quadrant. It will sometimes get a slightly larger rectangle because of the lines in between, which is dealt with later. It uses the angle of the quadrant to right itself, rotating the image to be horizontal. Then it goes through the same process of getting the white tablet pixels and producing a contour box around it. Then it crops the box into another image. If the image is not perfectly square, it will shave off some of the image. It converts the image to gray-scale, then thresholds it. This process is done to all four quadrants given.

Finally, the four quadrants are organized to form a full QR code. Because we do not know where each quadrant belongs, we run through the 24 permutations of each possible configuration. Each image contains a slice that is common with its neighbors. To combat this, a portion of some image is sliced off because we know the size of the slice compared to the whole quadrant. Then each grid of four is run through a QR recognition algorithm and tested for a four digit number.

Threat Identification and Behavior

In addition to general obstacle avoidance, it was imperative that the drone be equipped with a robust detection algorithm in order to combat the adversarial drones operating in the arena and safely avoid human operators. To this end, we implemented a deep neural-network based object detector with the tensorflow object detection API [4], capable of detecting both drones and humans at up to 30 frames per second. The final network architecture employed a variant of the Single-Shot Multibox Detector, or the SSD [5]. After the initial detection stage, the registered drones within the visual field would be tracked with the recent DaSiamRPN, winner of the VOT challenge 2018 with an open-sourced implementation on Github [6].

While there exists a wide array of available datasets taken from the aerial view of the drones, the object localization dataset of the drones themselves are relatively scarce. In order to combat the limited size of training data, a bootstrapping scheme was applied starting with a manually verified dataset and a YOLO [7]-based detector from [8]. Afterwards, a large collection of images from google images were collected with an automated script [9] with keywords based on the drone types, such as `quadrotor`, and the manufacturers, such as `DJI`. In order to sanitize the dataset, duplicate entries were culled with PHash [10], a perceptual hash algorithm with an implementation provided by OpenCV [11].

As a means to ensure proper inputs for the network pipeline, the resultant images were then manually verified. As manually generating bounding-box annotations can be cumbersome, especially with large-scale datasets, the annotation process consisted of a three-prong signal: `o` to indicate acceptable bounding boxes, `x` to indicate negative images, and `s` to indicate incomplete annotations that require operator intervention. The remaining annotations labeled `s` were manually inspected again, but the labor costs were greatly reduced to a smaller size of bounding-box annotations.

The resulting dataset was converted to a *tfrecord* format, and the detector was trained based on the tensorflow object detection API. The detector was trained (Figure 5) with an adapted architecture from the SSD network utilizing a PPN feature extractor configuration and focal loss. In order to save computational resources, a single detector was trained to detect both humans and drones; as a large corpus of pedestrian detection datasets are available, the dataset was merged with the MSCOCO [12] dataset where non-human entries were treated as negatives.

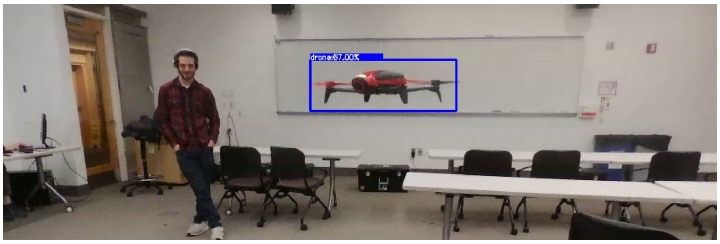

Figure 5. Drone Detection Result.

Despite considerable efforts to improve the performance of the detector, the results were never quite on par with the expected level of robustness for a drone operating in the same arena as humans. In order to boost the reliability of the detections while balancing precision and recall, the detection stage consisted of a two-step verification pipeline where only drones with high confidence values were initially selected. This set was then augmented with a second inference performed on cropped regions of the image from lower-confidence detections of the previous step.

In order to identify threats and obstacles more reliably and rapidly, we use three front facing ultrasonic rangefinders to aid in threat identification. This non-vision based approach is extremely robust, although short ranged, and ensures that we avoid collisions with both obstacles and enemy drones. We chose the LV-MaxSonar-EZ1 [13] ultrasonic range-finding sensors because of their ability to be chained as well as their wide field of detection (~50°). The three ultrasonics are each facing downwards by 20° to ensure the visual field is covered by the rangefinders, and are spaced by 45° horizontally ensuring some overlap (Figure 6).

When an object is detected by a rangefinder, its approximate location is added to a shared list of obstacle detections for all drones in the swarm to avoid. Any obstacle detections from the previous sensor update that are located within each rangefinder's field of view are removed from the list before the new obstacle detections are added. This step prevents drones from recording multiple sightings of the same obstacle and erases old obstacles that have since changed position.

After obstacles are identified by either computer vision or the onboard rangefinders, a potential-field gradient approach is used to simultaneously navigate the drone away from obstacles and towards its desired position [14]. At each point in time, the drone constructs a 2-dimensional potential field over the arena with a value corresponding to the desirability of being at each point. Any obstacles seen by the drone swarm correspond to lower potential values, with the decrease in potential inversely proportional to the distance from the obstacle. The entire field also slopes linearly upward toward the target position that the drone is trying to reach. The drone then moves in the direction of the gradient of the potential field at its current location, in order to maximize the increase in potential at any given time.

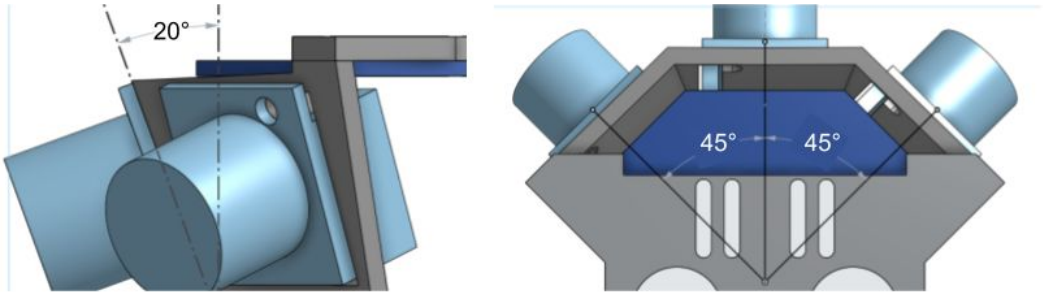

Figure 6. CAD of ultrasonic sensor placement, displaying mounting angles.

This algorithm is equivalent to treating each obstacle as a repulsive force with a value inversely proportional to the square of the distance, and adding a constant attractive force towards the drone's target position. The drone then moves in the direction of the sum of these force vectors. The attractive force and repulsive forces are scaled such that the drone will never approach closer than one meter to an obstacle. This computation is expressed in Equation 1, where $\vec{v}$ is the computed direction of motion, $\vec{x}$ is the drone's current position, $\vec{o}_i$ is the position of each detected obstacle, $\vec{g}$ is the drone's desired position, and r is the closest approach distance (one meter).

$$\vec{v} = r^2 \sum_{i=1}^{N} \frac{\vec{x}-\vec{o}_i}{|\vec{x}-\vec{o}_i|^3} + \frac{\vec{g}-\vec{x}}{|\vec{g}-\vec{x}|} \qquad (1)$$

This approach has the benefit of easily adapting to multiple obstacles while prioritizing those closest to the drone. The greatest risk is that the drone can get trapped in a local maximum of the potential field rather than the global maximum, but given the relative openness of the arena this was considered unlikely to occur, and furthermore operator voice commands can still be used to help the drone escape the local maximum.

Gesture Identification and Behavior

While most instructions can be more easily given via spoken commands, gesture commands provide an advantage when the human operator wants to specify spatial information with a greater precision than they can measure by their own observation. For this challenge, we used gesture commands to specify directions for a drone to move. Rather than requiring the operator to determine an exact numerical angle, they can instead point in the desired direction with their arm, which both increases precision and provides a more intuitive user interface.

The gesture recognition takes advantage of the voice interface by searching for a static gesture when a specific command is given rather than constantly scanning for motion. The specified drone takes an image with its camera, and then scans that image for the human's arm. To aid in finding the arm, distinctive colored patches are worn on the human's wrist and shoulder. Using the known value of the height of the human's shoulder above the ground, the distance from the drone to both the wrist and the shoulder can be computed via the geometric relationship given in Figure 7.

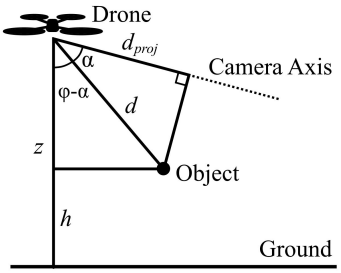

Figure 7. The geometric relationship between the observed position of an object in the camera image and the actual location of the object in space.

This relationship is defined in Equation 2, where φ is the camera pitch angle, h is the height of the point above the ground, z is the drone's altitude, d is the distance from the point to the drone, d_{proj} is that distance projected onto the camera axis, y_{cam} is the vertical position of the point in the camera image, and f_y is the camera's vertical focal length. The position of the wrist and shoulder in space can then be determined using their projected distances [15], and the vector from the shoulder to the hand provides the desired direction for the drone to move.

$$\alpha = tan^{-1}(y_{cam}/f_y)$$
$$d = (z - h)/cos(\varphi - \alpha)$$
$$d_{proj} = d\ cos(\alpha) \qquad (2)$$

Communications Systems

Our drones must communicate with the base computer, between each other, and also with the operator display. To deal with all of these channels, we chose to host our own wireless network for the drones to connect to. All of the Bebop drones, as well as the obstacle avoidance boards, are connected to this network and interface with a central computer which handles communication between drones and communication to the user. We use a wireless 2.4GHz microphone to communicate with the user, which connects directly to the auxiliary input port of the central computer.

Each of the three ultrasonic range finders communicates to an ESP8266 module on board each drone. They all connect to a server hosted by the central computer, and transmit their range information, which is then fed into the obstacle avoidance controller.

User Interface / Human-Machine Interface

A voice recognition interface is used to process commands from the human operator. For this task we use the CMUSphinx engine [16], implemented via the SpeechRecognition python library [17], because of its off-line functionality and its ability to define a custom syntax and vocabulary for interpreting commands via a Java Speech Grammar Format [18]. Limiting the possible command inputs in this way increases the ability of the algorithm to distinguish different commands and prevents invalid inputs from being processed. A wireless microphone is worn by the human operator to detect voice commands.

Each command consists of a target, an action, and any number of additional parameters (e.g. "swarm look north" or "alexa turn left 45"). The full command interface is shown in Table 2. A command's target specifies which drone or drones should act; commands can be addressed to either an individual drone by name or the entire swarm, and an unaddressed command will default to the same drone or drones as the previous command. Actions include both moving a given distance and turning, and in both cases the direction can be specified either relative to the drone's current orientation or using absolute cardinal directions. Distances can be given in meters, centimeters, inches, or feet, with meters as the default if not specified (e.g. "forward 5" is equivalent to "forward 5 meters"). Angles are always interpreted in degrees. As a safety precaution, the word "cancel" can be inserted at any point to invalidate a command.

TABLE 2. VOICE COMMAND INTERFACE

Target	*Action*	*Parameters*	*Desired Behavior*
swarm, alexa, google, siri, clippy	north/east/south/west	[number] m/cm/in/ft	Move in a cardinal direction
	forward		Move in the direction drone faces
	follow		Move in the direction of a gesture
	jump/duck		Increase or decrease altitude
	look	north/south/east/west	Turn to face a cardinal direction
	turn	left/right [number]	Rotate by a given angle
	takeoff/land	N/A	Takeoff or land
	stop	N/A	Hover in place
	picture	N/A	Take an image of a QR code

RISK REDUCTION

Vehicle Design

EMI/RFI Solutions

Bebop 2 Drones are off the shelf protected from EMI internally. In order to not add additional interference to the system, we chose to run our additional communication system for obstacle avoidance on WiFi using the ESP8266 modules. Additionally, we chose to use a frequency hopping radio controller and receiver in order to eliminate interference between drones for the EStop system.

Shock/Vibration Solutions

The Bebop 2 rotor chassis is isolated from the sensor suite by a vibration-dampening rubber system. When adding additional modules, we were careful not to interfere with these systems by

creating accommodating holes and cutouts for all moving parts on the bottom of the drone. The propeller guards also prevent damage to the drone by their flexible nature, which as been tested on the actual platform.

Safety

Propeller Guards

We designed fully enclosing propeller guards in order to make sure that a finger could not come in contact with the propeller blades. We designed these to be 3D printed from PETG since this plastic is especially resistant to shattering. The guards link together, forming a mesh case around the propellers. We verified that our flight time is not significantly reduced by the guards.

Speed Limitation

As mentioned in the discussion of the arbiter above, there is a maximum speed imposed on all of the drones, which prevents them from moving at dangerous speeds.

Modeling and Simulation

In order to expedite code development and testing without risking hardware damage or safety issues, we have developed simulations of varying levels of complexity: ranging from a Qt-based simple two-dimensional planar field model to a full Gazebo [19] simulation of the mission arena, as well as a drop-in replacement based on the Parrot-Sphinx simulator [20]. While a more abstract and simple 2D simulation was appropriate for testing logic-level code and strategy, the physics-enabled full 3D simulations allowed us to integrate our code with higher fidelity

Physical Testing

We have physically tested a mock up competition environment with the official bins several times, including recorded crashes with our prop guards. We verified that the prop guards do not shatter when the drone collides with walls, and also that the drones are able to respond to voice commands and follow high level commands. Much of our perception code has been tested live. Our person, gesture and QR detection algorithms have been tested live on drones several times, and bin detection has been tested with comparable videos.

CONCLUSION

We have created a drone swarm to aid a human operator through guided autonomy. The swarm is capable of autonomous flight, perceiving and avoiding threats, identifying and station keeping over vision targets, and responding to voice and gesture commands. The system has performed well in both simulation and physical testing, and the IARC 2019 competition will provide an opportunity to determine its effectiveness during a full trial of the mission.

REFERENCES

[1] Parrot Bebop 2. (2019, May 16). Retrieved May 31, 2019, from https://www.parrot.com/us/drones/parrot-bebop-2

[2] Rother, C., Kolmogorov, V., & Blake, A. (2004). "GrabCut". ACM SIGGRAPH 2004 Papers on - SIGGRAPH 04. doi:10.1145/1186562.1015720

[3] Zhu, Z., Wang, Q., Li, B., Wu, W., Yan, J., & Hu, W. (2018). Distractor-Aware Siamese Networks for Visual Object Tracking. Computer Vision – ECCV 2018 Lecture Notes in Computer Science, 103-119. doi:10.1007/978-3-030-01240-3_7
[4] Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., . . . Murphy, K. (2017). Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). doi:10.1109/cvpr.2017.351
[5] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C., & Berg, A. C. (2016). SSD: Single Shot MultiBox Detector. Computer Vision – ECCV 2016 Lecture Notes in Computer Science, 21-37. doi:10.1007/978-3-319-46448-0_2
[6] Foolwood. (2018, November 30). DaSiamRPN. Retrieved May 31, 2019, from https://github.com/foolwood/DaSiamRPN
[7] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). doi:10.1109/cvpr.2016.91
[8] Chuanenlin. (2019, May 25). Drone Net. Retrieved May 31, 2019, from https://github.com/chuanenlin/drone-net
[9] Hardikvasa. (2019, May 22). Google Images Download. Retrieved May 31, 2019, from https://github.com/hardikvasa/google-images-download
[10] PHash. (n.d.). Retrieved May 31, 2019, from https://www.phash.org/
[11] OpenCV. (2019, May 31). Retrieved May 31, 2019, from https://opencv.org/
[12] Lin, T., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., . . . Zitnick, C. L. (2014). Microsoft COCO: Common Objects in Context. *Computer Vision – ECCV 2014 Lecture Notes in Computer Science,*740-755. doi:10.1007/978-3-319-10602-1_48
[13] MB1010 LV-MaxSonar-EZ1. (n.d.). Retrieved May 31, 2019, from https://www.maxbotix.com/Ultrasonic_Sensors/MB1010.htm
[14] Khatib, O. (1986). Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. *The International Journal of Robotics Research,5*(1), 90-98. doi:10.1177/027836498600500106
[15] OpenCV. (n.d.). Camera Calibration and 3D Reconstruction. Retrieved May 31, 2019, from https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html
[16] Lamere, P., Kwok, P., Gouvêa, E., Raj, B., Singh, R., Walker, W., . . . Wolf, P. (2003). THE CMU SPHINX-4 SPEECH RECOGNITION SYSTEM. IEEE Intl. Conf. on Acoustics, Speech and Signal Processing. Retrieved May 31, 2019.
[17] SpeechRecognition. (n.d.). Retrieved May 31, 2019, from https://pypi.org/project/SpeechRecognition/
[18] Class JSGFGrammar. (n.d.). Retrieved May 31, 2019, from http://www.gavo.t.u-tokyo.ac.jp/~kuenishi/java/sphinx4/edu/cmu/sphinx/jsapi/JSGFGrammar.html
[19] Osrf. (n.d.). Why Gazebo? Retrieved May 31, 2019, from http://gazebosim.org/
[20] Parrot. (n.d.). What is Parrot-Sphinx. Retrieved May 31, 2019, from https://developer.parrot.com/docs/sphinx/whatissphinx.html