Autonomous Quadcopter for Multiple Robot Tracking and Interaction in GPS-Denied Environments

George Lachow, Alexander Khoury, Jason Quach, Eric Ho,
Ian Schroeder, Junru Ren
IEEE at UC San Diego
University of California, San Diego, La Jolla, CA 92093
georgelachow@gmail.com akhoury727@gmail.com j6quach@gmail.com erho@ucsd.edu
ian.schroeder@ieee.org j.ren@ieee.org

ABSTRACT

This paper describes the entire Unmanned Aerial Vehicle designed by IEEE at UC San Diego to compete in the International Robotics Competition. The system capable of interacting with objects, avoiding obstacles, within a GPS-Denied environment autonomously. Sensing the environment is strictly performed through a downward facing camera with a variety of sensors in order to establish positional estimates about the aircraft itself and the external agents. Immediately following these estimates an appropriate reaction is incurred. A majority of our system testing was performed in a software in loop simulator, enabling us to feasibly assess and evaluate a multitude of situations.

1 INTRODUCTION

Unmanned Aerial Vehicles (UAVs) were originally designed for military applications, but have expanded to many other applications such as commercial, recreational and surveillance. UAVs operate with a varying degree of autonomy, ranging from being piloted remotely by a human to piloted autonomously through on-board computing. While robust methods for object detection and avoidance are critical to this task, tracking and interaction with moving objects has not been proven thoroughly as of yet. Autonomy without external aids such as Global Positioning Systems (GPS) for positioning is becoming ever more paramount for indoor or military applications where GPS may be limited or unavailable. Mission 7 of the International Aerial Robotic Competition (IARC) [1] seeks to explore robust methods of interacting with moving obstacles while building the foundation for indoor localization.

In the first part of Mission 7 of IARC, Mission 7a, participants must develop a fully autonomous UAV to track randomly moving objects and interact physically with them in

order to guide them across the field to a designated location while avoiding dynamic obstacles. The indoor area consists of a field that is 20x20 meters, with grid lines every meter. The field is populated with 14 Roombas, 4 of which are obstacles with up to 2 meter vertical poles while the remaining 10 move in a semi-random pattern and have paddles on top which allow interaction.

The problem has 3 key main challenges, localizing the aircraft, detection and tracking of obstacles and targets, and optimized planning and control in order to complete the mission within a time constraint. Because of the clear problem definitions, we split up our solution into 3 main areas. The first is Self-Estimation, localizes the aircraft's absolute position relative to the grid. The next area of focus is Perception, identification and tracking of objects, along with determining the objects position relative to the aircraft. The third focus is Planning, which generates high-level trajectory commands to take to optimize herding.

This is the first time IEEE at UC San Diego has participated in IARC. Since Mission 7a has been active for a couple years, we were uncertain to what would be most challenging. With Mission 7b having a similar format, our goal was to attempt to solve part A, while building a strong foundation for the follow year. Building a future proof airframe was paramount for in the event of changing the electronics package. Also, we aimed to create a robust and accurate software in the loop simulator for easy development of our software package.

2 SYSTEM OVERVIEW

Our system is split up into two main components, consisting of a aircraft and a Ground Station (Figure 1.). The aircraft handles all low-level flight control processing on-board while also sending telemetry, sensor information and a video stream to the ground station. The information from the aircraft that is sent to the ground station is processed on a computer to determine the high-level flight instructions.

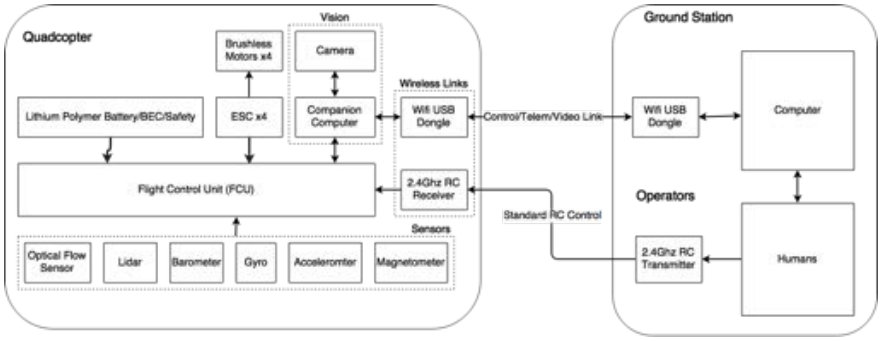


Figure 1: System Block Diagram

2.1 Quadcopter

Our aircraft consists of mainly commercial off the shelf components.. To ensure safe flight in the event of loss of communication, low-level flight controls are handled by a Pixhawk using

the Ardupilot flight stack. This is independent of the high-level flight instructions being sent from the ground station. An on-board computer receives flight instructions and sends the instructions to the Pixhawk over serial. The computer also receives the camera feed and telemetry data from the Pixhawk and sends it to the ground station.

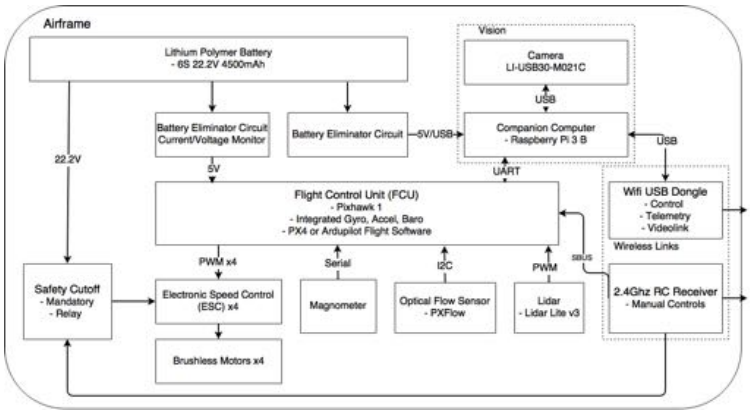


Figure 2: Aircraft Block Diagram

2.1.1 *Airframe*

The airframe chosen for mission 7a was a 640 class quadcopter. Because efforts were focused on software, a commercial airframe that included electronic speed controls, motors and propellers designed for light duty cinematography was chosen. This ensures a lightweight, well developed package designed for long flight times with additional payload.

2.1.2 *Propulsion*

Propulsion is achieved with four Hobbywing XRotor 40A ESCs powering T-Motor AntiGravity MN4006, 380kv motors paired with 15 inch Tarot 1555 foldable propellers. This motor and propeller combination can produce a maximum total thrust of 8.9 kilograms while consuming 360 Watts, far above aircraft weight. Although, these motors are extremely efficient at low RPM, consuming only 75 Watts at 50% throttle.

2.1.3 *Power Distribution*

The aircraft is powered by a 6 Cell, 22.2V 4500mah Lithium Polymer (LiPo) battery. The battery chosen was the largest watt hour pack, at 99.9Wh, we were allowed to bring on an airline. Without the sensor package, the vehicle's flight time is 35 minutes at 1.83kg. With the full payload, the operational flight time is an estimated 20 minutes. Two voltage regulators are on the aircraft. One is used to supply 5V to the companion computer, USB camera and high power WiFi dongle. The second is used to supply 5V, propulsion current draw and battery voltage to the flight controller.

2.1.4 *Flight Termination System*

Our Emergency Flight Termination was made possible through a modifying the IARC Common Kill Switch reference design. We had to modify the design because we have a higher voltage battery pack, at 28.2V peak compared to 18.8V, along with the motors capable 64 Amps compared to 35 Amps continuous. We added 3 additional N-channel MOSFETS to have a total of 6. We also had to replace the 5V linear voltage regulator with another 5V linear voltage regulator rated up to 28.2V.

2.1.5 *Sensors*

The Ardupilot flight control stack was originally built with GPS for holding a position. Since we cannot use GPS for positioning, holding a position in a hover is achieved with a PX4Flow optical flow camera which is solely used for the low-level flight control of the Arducopter flight stack and serves as a safety redundancy in the event of a loss of communication. The Pixhawk flight controller includes one barometer, two 3-axis gyroscope, two 3-axis accelerometers and a 3-axis magnetometer. Due to the built in barometer poor accuracy, we opted for lidar for altitude. We chose a LIDAR-Lite v3 for altitude which has a range from 0-40m and an accuracy up to 2.5cm. A down-facing camera with a wide angle lens is mounted to the bottom of the aircraft for vision needs.

2.1.6 *Communication*

There are two primary communication links from the aircraft to the ground station. The first communication link is primarily intended for triggering of the emergency safety switch but also serves as a method of switching the aircraft from manual mode to autonomous mode. This link is a standard 2.4GHz spread spectrum radio controller and receiver. The second communication link in our system supports all communications between the aircraft and the ground station computer, including video streaming, telemetry, sensor data and aircraft commands. This link is through a 802.11b/g/n WiFi Link using an USB WiFi Module with a maximum bandwidth of 150Mbps.

2.2 Software

The software component of the aircraft is built on a Raspberry Pi 3 running rasbian, with a ROS and MAVROS software stack to enable remote communication and live streaming to a nearby groundstation. The Ardupilot flight control stack handles low-level flight controls for determining flight path and control of the vehicle, while the majority of the computation is handled offboard via a nearby groundstation running Ubuntu 14.04 with MAVROS.

3 PERCEPTION

We perceive the environment entirely through a downwards facing camera with a large field of view. The problem in which perception solves is the detection and tracking of objects of interest within the arena. Here is where the computer vision suite resides, processing one frame at a time from the camera feed to solve these tasks. Additionally, the use of

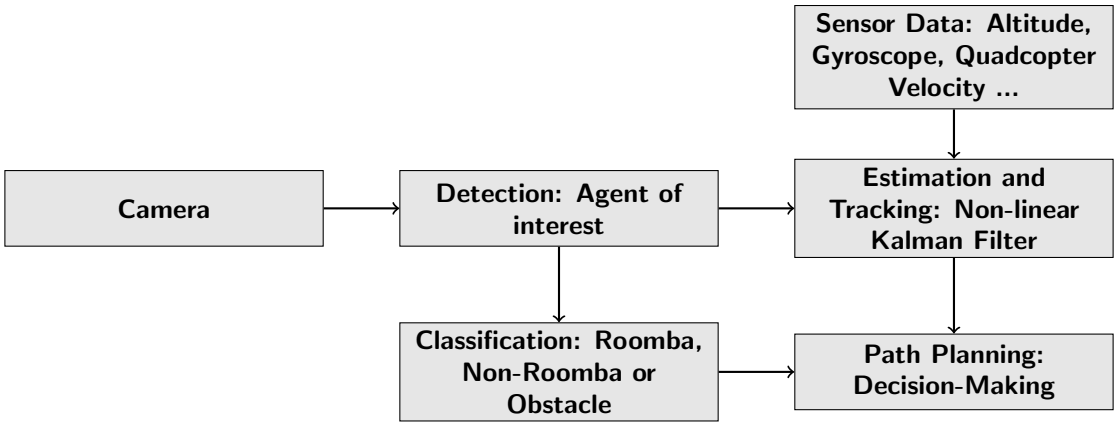


Figure 3: The computer vision pipeline to detect and track roombas

external sensory data from the Pixhawk enables additional information in which our tracking algorithm can utilize in order to develop more precise estimates for any particular object. The information calculated from each frame is than forwarded to our path planning and state estimation suites in order to react to and perform a variety of decision making tasks.

3.1 Detection and Classification

The roombas have very distinct colored features, with red and green paddles, therefore a threshold is performed over a range in the HSV colors-space to identify an agent of interest. This was preceded by a Gaussian Blur in order to remove small perturbations within the frame. Its pixel coordinates are than calculated by locating the center of its contour and a conversion is performed as explained in the next section.

However, a more robust and planned approach to solving detection is currently in progress. A color threshold alone can be effective in certain situations however more often than not lighting can perturb its defining feature resulting in poor detection performance. Therefore, we plan on collecting real world data with a machine-learning approach in mind to develop a more robust detector and classifier. Two possible sub-approaches for this task that we are testing involve hand-engineered feature extractions through SIFT descriptors and histogram of gradients with a support vector machine classifier or a neural network as an end to end algorithm of which learns the features itself from our dataset.

3.2 Position and Pose Estimation

A combination of both Pose Estimation and Linear Projection Estimation is utilized, operating on two perspectives of where an object is located in order to develop a more robust estimate. The use of a Kalman Filter based approach can fuse together these two estimates enabling for a predictive and stable holistic estimate for which state-estimation and path planning can operate on for decision making. This ultimately allows us to track multiple features of any particular agent, particularly the positional coordinates, velocities, and orientations.

3.2.1 *Pose Estimation*

The pose estimation stack is designed to allow for the tracking of all visible roombas in the frame. Before the corner features of the roombas can be extracted from the video feed, several preprocessing steps are taken. First, a Gaussian filter is applied to blur the image slightly, removing noise. Next, a multi-threshold algorithm is applied to isolate the roombas from the background. This step is also able to distinguish between the two colors of roomba on the field. Once the relevant features of the roomba are known, they are applied to the Perspective N-Point (PnP) algorithm to determine the roombas distance and orientation with respect to the video source. Since the intrinsic parameters of the camera are known in advance, this is a fairly simple step. The parameters are determined as follows:

The Pose Estimation node. When combined with the orientation of the camera relative to the field (more math), the angle at which each roomba needs to be moved to orient with the goal line is known to the aircraft and can be utilized in flight planning.

3.2.2 *Linear Projection Estimation (LPE)*

Given the constraint of using a down-facing camera, the center of the camera acts as the aircraft's origin relative to the arena. However, we must account for the aircraft's pitch and roll, where the center of the camera is not facing perpendicular to the floor. Therefore, a readjustment of the projection of the aircraft's origin is performed by operating on gyroscopic data to re-project this origin. After obtaining the aircraft's origin lines are drawn between all detected roomba origins to the aircraft's origin in pixel coordinate values. A conversion from pixel to world coordinates must be performed in order to calculate the true relative position. Using altitude data, we can rescale the line magnitude to correctly determine the distance relative to the aircraft in world coordinates. This is a computationally quick estimate and can operate within the means of an embedded device with little expense.

4 PLANNING

4.1 Roomba Following

Once our planning algorithm chooses a roomba to "score," it moves towards the roomba, and follows the roomba from above. This following algorithm utilizes two Proportional-Integral-Derivative controllers(PID), one for Roll and the other for Pitch. Using the roomba position information output by the perception stack, we are able to calculate the PID coefficients and implement the controllers. By tuning the PID gains, we are able to adjust the response of the aircraft to a change in position of the roomba, resulting in a smooth and accurate following of the roomba.

Figure 4: Quadcopter following a roomba in our simulation

4.2 Roomba Landing

The landing algorithm immediately proceeds the "Following" algorithm, where once we believe we have an accurate "follow" of the roomba, our quadcopter can slowly descend onto the roomba, pushing the button. We then increase our altitude until we can detect the state and heading of the roomba, such that either we land once more or return to a follow, such that it moves towards the goal line.

5 SELF-ESTIMATION

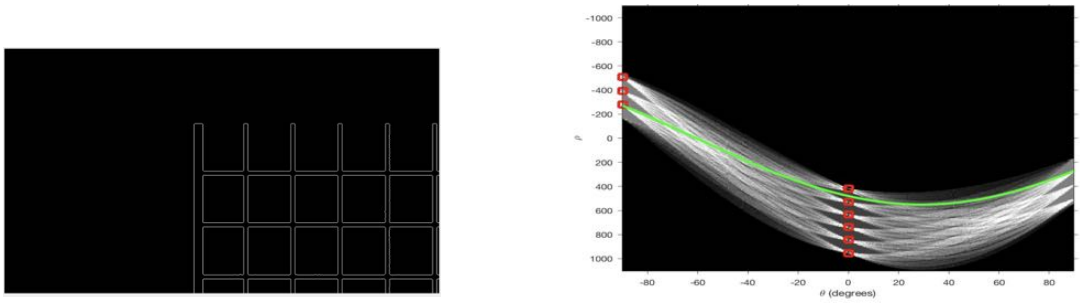
The UAV depends on information in its surroundings such as visual location and its relative orientation. In this paper, we present a solution to self-localization that consists of kinematics and traits of the hough transform. We can exploit the uniformity of the 20m by 20m square grid being compromised of smaller 1m by 1m squares.

5.1 Grid Detection

Grid detection is made possible through a few steps. First, we apply a white threshold to highlight the grid lines (Figure 5.a). Then we apply a canny edge detector to identify the edges of the lines. Next, a hough line transform is done in order to find the location and angles of the grid lines.

5.2 Localization

Since we identified the location and angle of the grid lines from the hough transform (Figure 5.b), we can exploit the hough space. The green sinusoid represents the center the camera image, where the red squares are the grid lines. In particular, in this case, three horizontal grid lines are represented by the three red squares marked at the -90 degrees; six vertical lines we observed in the real world are the six red dots marked at the 0 degrees. The green sinusoid represents the center of the image in the hough space. The green curve passes through the third red square from top at -90 degrees, which implies that the third horizontal line passes through the center of view; similarly, the green sinusoid is between the fifth and sixth grid line, showing that we are between them.



(a) Grid after filtering and edge detection

(b) Hough line transformation of Grid w/ UAV Position

Figure 5: Grid Detection and Localization

As the UAV starts moving over the grid, both horizontal and vertical grid lines would pass through the center of view. In Hough Space, the red squares (lines) would cross the green sinusoid (center of view). This is achieved by tracking the two closest a red squares relative to the center of view. During movement, a red square crosses the green sinusoid indicating the direction and the change in position of the UAV. Through the use of the inverse hough transform, we can also localize ourself inside a grid cell. Using this, we can localize ourselves relative to our location on the grid by counting how many grid lines we traversed, both vertically and horizontally.

5.3 Kinematics

Using our gyroscope and accelerometer, we are able to transmit the UAV's yaw data and its velocity to the ground station. To map the UAV's coordinates to a map of the arena, we calculate it's x and y position relative to the starting position of the UAV. Assuming a sample rate t, we can calculate the total distance

$$d = v * t$$

where v is its velocity. We then can find it's x and y coordinates using the equations

$$x = d * cos()$$

and

$$y = d * sin()$$

where is the gyroscope's yaw data. We simulated this kinematics method using Matlab as shown in Figure X. In the simulation, we set the origin to be the center of the arena and we also displayed the trail of the UAV. Since this method accumulates error over time, it serves as a sanity check to the method described in the previous section.

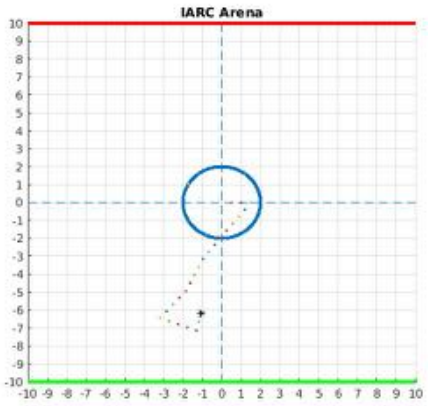


Figure 6: IARC Arena

6 SIMULATION

In the absence of a physical or finished aircraft, and for safety reasons, initial testing needed to be done on a simulation. We developed a software in the loop simulation in order to develop our software package. The aircraft's software stack exists within the ROS framework, which integrates directly with the 3D simulation software, Gazebo.

6.1 Gazebo

Gazebo allows numerous customizations to be done to allow incredible flexibility in robot simulation. It also supports generation of measurements from a multitude of sensors, allowing customization for additive noise. Gazebo provides extensive documentation [2] and examples of how to implement and simulate various situations. They provide direct access to their API with the use of plugins, allowing for development of control software for robots, sensors and even the environment.

6.2 The Reference

The development process for our simulation began by referencing the implementation achieved by Aerial Robotics Kharagpur [4]. Their implementation served as an excellent starting point for our team; however, there were many improvements that could be made to allow for a more realistic, more accurate simulation.

6.3 Improvements

6.3.1 *Floor Texture and Lines*

Firstly, improvements on the gazebo world were made, namely the arena floor. We replaced the texture of the arena floor, increasing its resolution. We also redrew the grid to incorporate the double thick line in the center, as well as the colored goal lines.

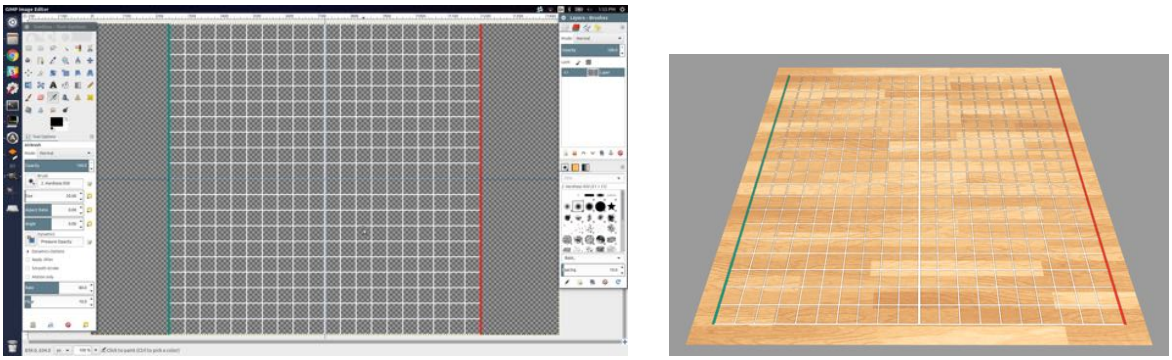


Figure 7: Drawing the arena lines with GIMP (Left) and Empty field with new floor texture and new lines in Gazebo (Right

Figure 8

6.3.2 *Roomba Models*

Next, the vanilla gazebo model for the iRobot Create, provided in the gazebo models library, needed customization to match the mission 7 aesthetic. Thus, using sdf, a derivation from XML specialized for describing robot models and environments, we were able to adapt the roomba models to feature a colored top plate, and a touch paddle more consistent with mission 7.

Figure 9: Original Roomba Model (Left) Updated Roomba Models (Center and Right)

Also, the roombas designed to be obstacles in mission 7 were also modified, adjusting the size of the 'PVC' attachment to match dimensions consistent with the rulebook.

6.3.3 *Quadcopter Model and Sensors*

Another area needing improvement is the quadcopter model. The included model comes from the hector quadrotor package provided by ROS. This model is more simplistic than we would have liked, as it does not account for the complicated dynamics and movement of a real quadcopter. It also is not compatible with MAVROS, which links ROS to the MAVLink communication protocol used on most physical autopilot systems. Therefore, we decided to replace it with the Erle-Copter.

Figure 10: Original Quadrotor Model (Left) and New Erle-Copter Model (Right)

Following the Erle-Copter simulation documentation provided by Erle Robotics [3], and with considerable tweaking, we were able to add the Erle-Copter model to our simulation. The Erle-Copter's movements are incredibly realistic, as it accounts for the quadcopter's dynamics. The model is realistic looking, with spinning rotors, and a high resolution model. It also features the prized compatibility with MAVROS/MAVLink. Basic movement commands may be sent to the Erle-Copter via the MAVProxy console, or more complicated commands programmatically by overriding the RC input to the quadcopter.

Some of the many sensors included with the Erle-Copter are a forwards facing lidar and camera, and a downwards facing camera. The Erle-Copter model information is written in urdf with xacro, much like sdf it is in XML syntax. We modified the model to better match what we expected for our physical quadcopter, removing unnecessary sensors, and adding a downwards facing camera (optional fisheye image feed), a downwards facing lidar (for accurate altitude measurements), and an optical flow sensor.

6.4 Current Simulation

Our simulation is an amalgamation of various packages, working seemlessly to deliver an aesthetically pleasing, highly functional, easily customizable, simulation environment for ROS software development. It integrates the Ardupilot SITL plugin, MAVROS, MAVLink/MAVProxy, ROS/GAZEBO, and many other packages, including our own software stack.

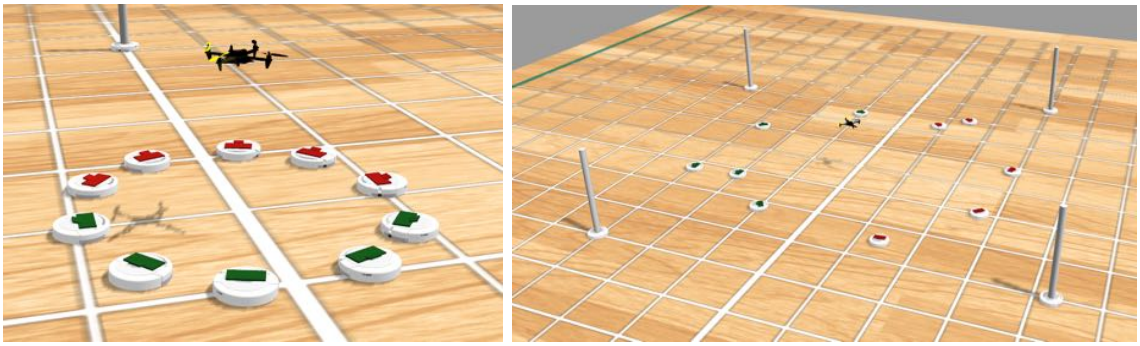


Figure 11: The UCSD IARC Simulation for Mission 7

Because gazebo communicates and integrates with ROS directly, sensor and robot information being published can be accessed via the ROS topics. Therefore, for example the camera image feed information can be accessed at any time.

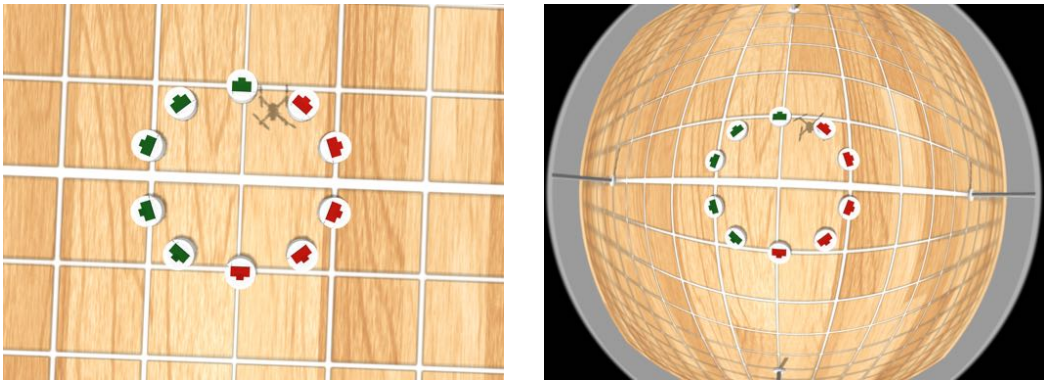


Figure 12: Downwards facing camera (Left) with fisheye lens (Right)

7 CONCLUSION

The challenges presented in Mission 7a are not only challenging from a software perspective, but also from a system integration perspective. Our solution involved integrating multiple complex systems while working with of open source software packages and commercial hardware. testing the system was a logistics challenge, building a robust software in loop simulator that allows us to test our software package on a nearly identical aircraft model within an environment modeled after the IARC arena was paramount. Abstracting the challenges into well defined packages allowed us to streamline development process. Our object identification and trajectory estimation works well within our simulation, but we are planning on improving its robustness in the future. We are currently still in the process of finalizing our Localization algorithm, which has performed well in testing. The next step for us would be to develop an algorithm for herding the obstacles since currently we can only follow a roomba and change its direction. We wanted to prove robust localization and object identification and tracking before tackling the herding optimization problem and autonomously changing strategies mid-flight.

REFERENCES

[1] I. A. R. C. Official Rules for the International Aerial Robotics Competition. `http://www.aerialroboticscompetition.org/downloads/mission7rules_013017.pdf`.

[2] Gazebo. Gazebo API Reference. `http://osrf-distributions.s3.amazonaws.com/gazebo/api/7.1.0/index.html`.

[3] E. Robotics. Gazebo Simulation. `http://docs.erlerobotics.com/simulation`.

[4] A. R. K. Team. Kharagpur Quadrotor Simulator. `https://github.com/quadrotor-IITKgp/quad_simulator`.