Autonomously Herding Ground Robots in a GPS-Denied Environment

MIT UAV Team*
Massachusetts Institute of Technology
Cambridge, MA 02139
mit-uav-core@mit.edu

Abstract

This paper describes MIT UAV Team's full system for Mission 7a of the International Aerial Robotics Competition (IARC). Our system involves a hardware layer and vision, modeling, communication, and planning software layers to herd Roombas in an environment denied of external navigation aids such as GPS or large stationary points of reference. The vision component processes the camera input from a GoPro attached to the aerial vehicle to identify Roombas and gridlines at each frame. These partial observations of the field are combined to create a global model, which allows for vehicle localization and Roomba tracking. Using this model, our vehicle utilizes a heuristic strategy for high-level coordination planning consisting of a finite state machine, where each state is a hard-coded "behavior module".

1 Introduction

The seventh mission of the IARC involves interaction between an aerial vehicle and moving ground robots. In particular, Roombas moving in a noisy trajectory must be guided across one side of a square field within a time constraint all while avoiding four moving obstacles in the field. The interaction of an agent with moving objects in an environment has applications in the use of aerial robots in moving platforms such as ships, trucks, or other air vehicles.

There has been extensive previous work on aircraft control and navigation in noisy environments [6, 5, 1]. However, few of these works involve the aerial robot interacting closely with moving objects in the environment. [4, 3] In such cases, determining the position of the aircraft is crucial.

Traditionally, the aircraft position is determined using a combination of a global positioning systems (GPS) receiver and an inertial measurement unit (IMU). However, this solution is prone failure when the GPS signal is weak or unavailable, as in indoor environments. In such cases, the localization task is handled with a vision system combined with other on-board sensors.[2] These methods, such as Simultaneous Localization and Mapping (SLAM), generally rely on large stationary points of reference such as walls.

In our work, we are constrained with an environment denied of all external navigation aids except for gridlines on the field. As such, we introduce a localization method based on gridline tracking which allows the aircraft to build a global model of the field from frame-level observations. Our localization and mapping method is combined with a heuristic strategy for high-level coordination planning to complete Mission 7a of the IARC.

In the following subsection, we describe this mission in more detail. In the next section, we give a high-level systems overview of our approach. In Section 3, we further describe the four main

*Authors: Martin Schneider, Simanta Gautam, Anthony Liu, Sam Udotong

subsystems. In Section 4, we include our results on a 3-D simulation of the mission created in Gazebo. The conclusion and future steps are discussed in Section 5.

1.1 Mission 7a

The arena consists of a 20-by-20 meter field with three red sides and one green side. The entire field contains gridlines that divide the field into 400 one-by-one meter squares. The UAV hovers near one of the corners of the arena at the start of the game. During the mission, it cannot hover above three meters from the ground nor leave the arena for more than five seconds.

There are 10 Roombas initially placed around the center of the arena facing outwards. Every 5 seconds, a trajectory noise of $\leq \pm 20$ degrees is added independently to each Roomba, and every 20 seconds, all of the Roombas rotate 180 degrees. Otherwise, the Roombas move in a straight line at 0.33 meters per second.

In addition to these 10 Roombas, which the UAV must herd, there are also four obstacle Roombas with large poles of varying length on top that the UAV must avoid. The obstacle Roombas are placed 5 meters from the center, and all travel in clockwise in a fixed circular motion. Figure 1 depicts the arena near the start of the game.

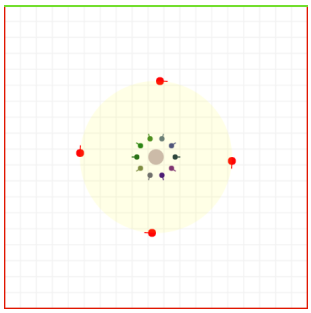

Figure 1: Simulated top view of the arena at the start of the game. The four obstacle Roombas are shown as red circles, the 10 regular Roombas are closer to the center facing outwards, and the UAV is hovering over the center of the arena.

The mission consists of autonomously herding the 10 Roombas with a UAV across the green side of the arena. The UAV acts on a Roomba by tapping the top of the Roomba to move it 45 degrees clockwise. Additionally, the UAV can also land in front of a Roomba, which activates the Roomba's collision sensor, turning it 180 degrees. The task of the UAV is to herd at least seven Roombas across the green side of the arena within 10 minutes.

2 System Overview

The full Roomba herding system can be cleanly divided between hardware and software and then further divided by the different internal mechanisms. The hardware system is a modified 3DR Solo Drone equipped with a camera and enhanced with additional optical flow and rangefinder sensors. Through these onboard sensors, the drone perceives the environment, builds a model, then acts on the environment to solve the task. The software is broken down into four general subsystems: vision, modeling, planning, and communications.

Vision and modeling work in tandem to construct an immediate belief model and a belief model over time, respectively. Planning then chooses a particular behavior among several hard-coded behaviors, along with a sequence of actions dependent on the current behavior. It utilizes the belief model constructed by the modeling component. Finally, the communications part consists of the Mission Executive component and the flight controller. The Mission Executive component is necessary for filtering the sequence of actions to be sent to the flight controller for obstacle avoidance, tapping, and validating that the drone is within arena boundaries. Our system, with the discussed components, is shown in Figure 2.

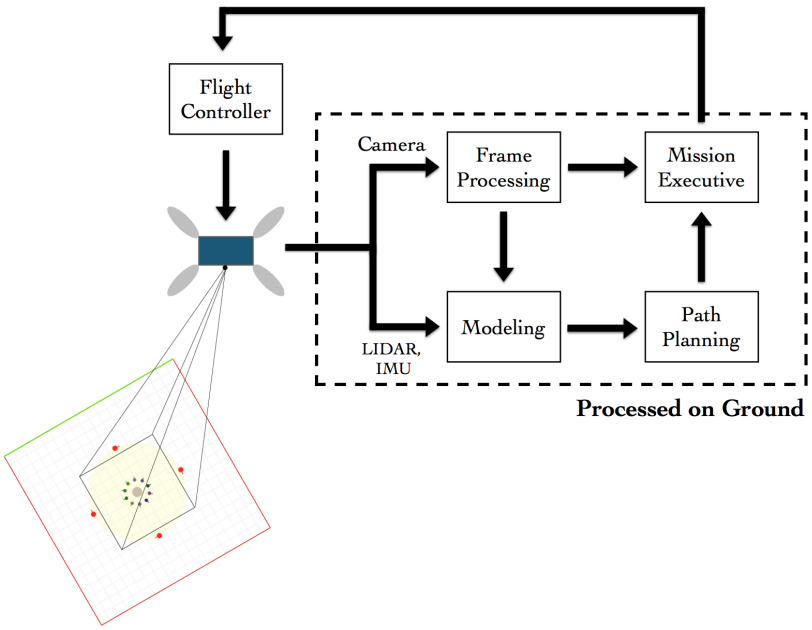

Figure 2: A diagram depicting the drone perceiving the environment, constructing an internal representation, the model, of the environment, and executing a planned path. Processing is done end-to-end on ground, mapping observations of the environment from onboard sensors to actions.

3 Subsystems

3.1 Hardware and Communication

The hardware and communication systems used for this project is based off the 3DR Solo Drone. Rather than building a custom hardware and communication solution, we opted to use the Solo based on reliability of the Pixhawk, strength of the open-source community, and familiarity with ArduPilot, the autopilot firmware. In addition to the Solo Gimbal and GoPro, we equipped the Solo with a LIDAR Lite optical rangefinder, a PX4Flow sensor, and a kill switch. The design decisions and interactions between these hardware components are discussed in the following section.

The flight controller used onboard the Solo is the latest stable developer release of the quadcopter version of ArduPilot – ArduCopter PX4-quad version 3.4.0. The Solo uses a Pixhawk 2 to run ArduCopter onboard. The Mavlink protocol is supported by this software, which is used to send and receive information between the groundstation and the quadcopter's autopilot. Through Mavlink, data such as the quadcopter's current yaw angle, it's arming state, and it's local position are accessible. Additionally, commands can be sent through Mavlink to instruct the quadcopter to takeoff, land, navigate to a position, and more. The open-source ArduPilot community is very active, and this year was particularly advantageous to use the Solo as much work was done on GPS-denied navigation.

The PX4Flow sensor can be seen as a hardware substitute in GPS-denied environment. It is an optical flow sensor that tracks the change in pixels of a point and estimates velocity. This sensor reading is a substitute for the velocity received from a GPS signal, and allows the quadcopter's onboard estimate of position to converge. Coupling the optical flow sensor with the LIDAR Lite optical rangefinder allows for a precise measurement of height that feeds into the velocity and position estimates. These two sensors enable precise indoor navigation of the Solo and reliable autonomous flight.

The kill switch was built to take advantage of the 3DR Solo's electronic speed controller (ESC) functionality. The Solo's ESCs automatically cut power to the motor as soon as the signal wire is cut. This is important, as we are able to cut power to the motor without needing to account for the total

80A of current that can be sent to the motors at maximum. Our design of the kill switch enables us to cut all four signal wires to the motors at the same time, leading to an instantaneous cut to the motors' power. These signal wires are sent through a relay that has a separate remote controller. This system is completely independent of the groundstation and on-board autopilot. Two remote controllers are provided and are on a different radio frequency than any other component onboard which eliminates radio interference.

A horizontal plexiglass plate is attached on the bottom of the quadcopter attached to the legs of the Solo. This custom plate is intended to be the piece of hardware that activates a Roomba upon landing. A hole is cut for each of the three optical elements on the bottom of the Solo: the optical flow sensor, the optical rangefinder, and the downward facing GoPro camera.

3.2 Vision and Modeling

The vision system
is the first level of processing. It takes in the raw camera input and outputs a list of gridlines and Roomba positions represented in pixels. The transformation happens over a number of stages using a mixture of OpenCV functions and custom image processing code. A GoPro Hero 3 attached to the bottom
of the drone using a Gimbal provides a continuous downward facing camera stream. The first stage of vision processing reads in the GoPro image and debulges it to remove warp effects from the GoPro's large field of view. The debulging is accomplished through a spherical remapping tuned to the GoPro's focal length, which produces a resulting image with straightened gridlines.

Now that the camera feed is straightened,
image processing begins by identifying Roombas. The image is thresholded by red and green, noise is removed, and connected components are identified. The connected components are then pruned to identify potential Roomba candidates (and remove other similarly colored objects, such as gridlines). The centers of the potential Roomba candidates are calculated, and saved.

The Roomba
identification is placed on hold and the gridline identification begins. The image is thresholded to produce a binary image representing the white areas of the camera frame. The binary image is then cleaned to remove noise from white specks that aren't a part of the gridlines. Using a Hough line transform, the image is then fitted with a number of lines represented by starting and ending pixel positions. Over the remainder of the processing steps, these lines will be transformed to be represented by their line number on the field.

Dealing with the camera frame directly is now complete, and line processing & pruning begins. The primary axes are identified. Duplicate lines are then merged using information about the line's starting/ending points and angles. A number of pruning steps then occur: short, misoriented, floating, and intermediate endlines are all processed out to deal with a number of edge cases. Endlines are determined by looking for lines that act as boundaries which no other lines extend past. The pruned lines are then reprocessed using the new endline information, and incorrectly removed lines are re-added. We now have a set of lines represented in pixels each of which corresponds to a distinct gridline in the camera frame.

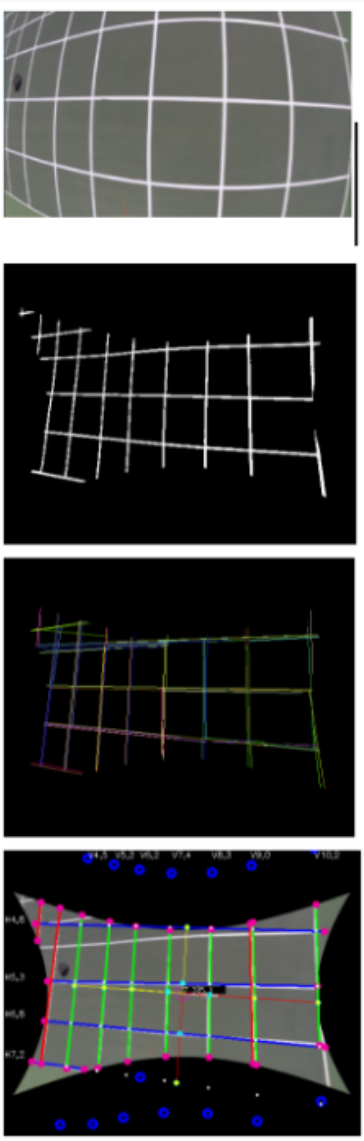

Figure 3: Representation of vision processing stages in order

The modeling system deals with the next level of processing:
transforming vision's gridline & Roomba estimates into continuous
position estimates for the UAV & Roombas. Most of the modeling processing happens closely in tandem with vision, with data and processing steps being traded back and forth.

Modeling takes vision's gridline estimates and compares them to gridline estimates from the previous frame. Additionally, modeling maintains a few persistent variables: the UAV's integer coordinate position, the UAV's floating point coordinate position, and the current axis of rotation. Combined, this information allows for a persistent beliefs about gridline numbers and UAV positions.

The current
gridlines and previous gridlines are compared and greedily merged. The center of the camera frame is considered to be directly below the UAV: if a line was previously on one side of the center and is on the other after the update, the believed UAV's integer coordinate position is updated accordingly. The line numbers are then updated based on the new belief of the UAV's position. A corrective step occurs to check that no unreasonable line numbers are predicted (lines that would be off the grid) and to ensure that the endlines identified from vision have the correct numbers. Finally, the UAV's floating point coordinate position is updated by comparing the distance to the immediate gridlines on the UAV's right and left.

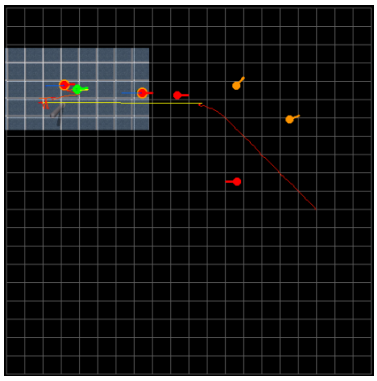

Figure 4: UAV & Roomba Model

The Roomba model persists and updates information
about Roomba beliefs across frames. Game rules are used in the update step: Roombas move at a consistent velocity and turn around every 20 second. Each Roomba is represented by a coordinate position, an angle, and a belief certainty that decays over time.

Modeling takes vision's identified Roombas in the camera feed (represented in pixels) and transforms them into grid positions (in the same manner that the center of the camera is transformed into the UAV's floating point coordinate position). Next, modeling updates the Roomba positions by greedily merging the observed Roombas with Roombas from the model. Missing Roombas that should be in the camera frame (but aren't) are pruned.

3.3 Path Planning

Vision and modeling act as the UAV's peripheral system; they read in sensor input and translate it to a believed view of the world. Planning takes this model and treats it as a reality. As the final level of processing, the planning subsystem directs the communication system where to move the UAV to on the field.

Initially, we attempted to apply deep reinforcement learning to the planning problem directly. We provided a deep RL agent with the true state of a simplified, two-dimensional simulation, and we rewarded it when it successfully tapped Roombas and when it scored points. We heavily penalized crashing into obstacles and leaving the arena.

With these rewards in place, the agent quickly learned to stay within the bounds and to avoid the obstacle Roombas. It eventually learned how to tap Roombas. However, at this point, learning plateaued, and the system did not improve further. Perhaps this was due to our reward function being too vague, but regardless, we sought other strategies.

We made a user-controlled, two-dimensional simulation to examine how humans would perform if they had full control of the UAV. We soon discovered that we could consistently beat the challenge (score seven Roombas) with a little bit of practice. Having had this experience, we realized that human players adopt a handful of distinct strategies at different stages of the game. These strategies complement one another in such a way that makes successful play possible.

A number of such strategies were developed and then tested in our Gazebo simulator.

The "follow" strategy “babysits” the topmost (closest to the green line) Roomba in the Roomba model. This refers to following a Roomba and tapping it whenever its path deviates from that of the direct route to the goal line. Humans use this strategy when they mentally commit to scoring a particular Roomba.

The "circle" strategy loops around the arena to get a sense of where all of the Roombas are. After all, only a fraction of the arena floor is visible at any given time.

The "defensive" strategy travels along the out-of-bound edges and taps Roombas that are in danger of leaving. This helps prevent potential future points from being lost and is an essential part of scoring seven or more Roombas.

Our planning strategy involves alternating between these strategies when certain criteria of the belief state are met. Time is an essential component of the transition decisions, but the locations and angles of the roombas also matter. Note: planning controls both the movement of the UAV and when it decides to land and tap a roomba.

3.4 Tapping

Tapping roombas involves a coordinated effort between the vision, modeling, and comm systems.

Vision operates in two modes: the “observing” mode described above and an additional “landing” mode. The secondary mode is used when tapping to ensure that estimates don’t get confused when the UAV descends. In landing mode, vision disables position processing/updates and switches to tracking the target’s pixel position, which is communicated to the communication system.

The communication system executes tapping by attempting to keep vision’s reported pixel target in the center of the camera feed. This is done by linearly adjusting the speed based on the distance of the pixel target from the camera’s center.

4 Simulation Results

Our end-to-end, three-dimensional simulation of mission 7a incorporates all of our major subsystems: vision, modeling, planning, and comm. We simulate a camera in software, we update our belief state of where all of the roombas are, and we query planning to determine the next action to take.

Our system has performed well in our simulations. We’ve been able to score 7-9 roombas reliably, and each component of the system is working as expected. Due to the additional unconsidered factors of noise and latency, is unlikely that these numbers will be replicated in a full real hardware test with the current system.

5 Conclusion and Future Steps

In summary, mission 7a is a difficult hardware, computer vision, modeling, and path planning problem that requires the integration of multiple complex systems to successfully complete the challenge. By identifying and abstracting these four components out of the problem, we tackled each separately, connecting their inputs and outputs in one final step at the end. This proved to be an effective approach towards this mission, and it led to successful performance in simulation.

As a future step, we would like to improve the path planning subsystem. Currently, the transitions between the behavior modules are handcrafted. We could instead use reinforcement learning [7] to learn a transition policy on the aircraft *behavior*. Abstracting the actions to a high-level behavior transition rather than a low-level motor control allows for a reasonable state space to learn an optimal policy. This approach would be a good combination of an expert system, where coordination strategies are hand-engineered, and a statistical system, where coordination strategies are learned from experience.

More ambitiously, another future direction is for the aircraft to learn coordination strategies directly with reinforcement learning. This would require (1) an incredibly apt reward function in order to steer learning towards viable modes of play and (2) deep neural networks for function approximation, as the state space would be very large. A deep reinforcement learning approach, while

computationally expensive and potentially an overkill for the challenge, would greatly expand the potential of the UAV agent in terms of the complexity of its strategy.

In concert, such an advancement would suggest a form of hierarchical deep reinforcement learning ; we would learn both the modes of play and the transitions between them. Although we tried using deep reinforcement learning to solve the entirety of the planning challenge, we didn't segment the problem in this way. This formulation of the problem to the RL agent could enable it to find a suitable solution in the now smaller state space.

One might also imagine ways to make the problem more difficult. The predictability of a Roomba's random motion makes this problem far easier for two reasons. One, it takes a few minutes for Roombas to begin leaving the arena on their own, and two, it allows modeling to make some assumptions about the state transitions. An interesting change might be to alter the form of random motion at a random point in the game. This would require the agent to intelligently adapt to a new strategy on the fly, suggesting a much. Mission 7a withheld from this level of uncertainty, but there's a lot of potential for for these types of challenges, and of course a lot of potential real world applications in the same space.

Acknowledgments

We would like to thank our club advisor Jonathan How for helpful discussions with path planning. Additionally, we would like to thank Suresh Kannan for guidance and feedback on our methodology and for steering us in a more productive direction.

We are also grateful for 3DR for providing us with the Solo aircraft, and MIT Edgerton Center for providing us with a great workplace. In addition, we would like to thank Airware, MIT CSAIL, and MIT EECS and Mechanical Engineering Departments for their sponsorship towards this project.

References

[1] C Goerzen, Z Kong, and B Mettler. A survey of motion planning algorithms from the perspective of autonomous uav guidance. *Journal of Intelligent and Robotic Systems*, 57(1):65–100, 2010.

[2] Pascual Campoy Juan F Correa Iván F Mondragón Carol Martínez Miguel Olivares Jorge Artieda, José M Sebastian. Visual 3-d slam from uavs. *Journal of Intelligent and Robotic Systems*, 55:299–321, Springer Netherlands.

[3] Damon Gerhardt Joseph L. Cooper Morgan Quigley Julie A. Adams Michael Goodrich, Bryan S. Morse and Curtis Humphrey. Supporting wilderness search and rescue using a camera-equipped mini uav. *J. Field Robotics*, 25:89–110, 2007.

[4] Stephen Griffiths Andrew Eldredge Randal W Beard Morgan Quigley, Michael A Goodrich. Target acquisition, localization, and surveillance using a fixed-wing mini-uav and gimbaled camera. *Robotics and Automation*, pages 2600–2605, 2005.

[5] Arthur Richards, John Bellingham, Michael Tillerson, and Jonathan How. Coordination and control of multiple uavs. *AIAA guidance, navigation, and control conference*, 2002.

[6] Arthur Richards and Jonathan How. Aircraft trajectory planning with collision avoidance using mixed integer linear programming. *American Control Conference*, 3:1936–1941, 2002.

[7] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, volume 135. MIT Press Cambridge, 1998.