Intelligent Quadcopter for IARC 2018

Noel Brownback
The University of Texas at Austin
Anthony Carreon
The University of Texas at Austin
Shayer Hassan
The University of Texas at Austin
Eric Johnson
The University of Texas at Austin
Colin Lewis
The University of Texas at Austin
Mark Loveland
The University of Texas at Austin
Umer Salman
The University of Texas at Austin
Ethan Starr
The University of Texas at Austin

ABSTRACT

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

INTRODUCTION

Previous aerial robotics missions have always allowed for GPS or set navigation points that would allow for constant location data for the drone. Location data is important to any mission because awareness of this information is the foundation to completing the given mission. Lack thereof navigation data could result in various failures, including complete loss of control of the drone. On top of this, the mission is to also design the dorne around the fact that the targets the drone is trying to herd are ground targets. The drone must touch the targets to navigate the targets, rather than the usual pick up and move to the destination.

The solution we came up with to address these challenges was enacting the use of computer vision as the basis to the drones function and execution of tasks. Through computer vision the bot has the ability to both understand its location, and also track targets in real time.

The drone records the data and position of the targets, and then for targets out of sight simulates the position of the targets until they come into sight, which allows for strategic analysis and definition based on estimates of unseen/out of sight targets.

Yearly Objectives: 2017 - Have a working bot, with computer vision software in working or nearly working 2018 - Continue development of computer vision, develop a computational game strategy through creation of simulations of the motion of roombas, tweak the physique of the drone for better flight control and stability

Air Vehicle

Propulsion and Lift System

The vehicle utilizes a x-y axis symmetrical quad-rotor system. This provides a balanced airframe optimized for omni-directional maneuvering. The frame is lifted by four T-Motor U3 KV700 motors. These motors provide 1.8 [kg] of thrust per motor at 100% throttle when paired with a 4-cell battery and 13 [in] propellers for total 7.2 [kg] of thrust. This is necessary for our 3.15 kg weight to maintain a 2.2 thrust-to-weight ratio ideal for battery life and flight time. The propellers used with the motors are 13 [in] long and made of carbon fiber. These T-Motor propellers provide excellent precision, durability, and efficiency. The rotors are placed to counter-rotate for flight stability and to retain a one-to-one ratio of propellers to motors.

Guidance, Navigation, and Control

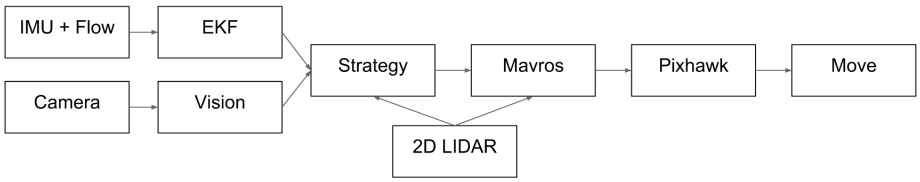
Stability Augmentation System

Our system builds off of the robust open source project Arducopter. Arducopter, runs on the Pixhawk, and controls the stability of the drone by taking in data from the IMU, compass, altitude LIDAR and the optical flow sensor. The Arducopter flight stack is maintained by hundreds of developers from around the world and the software is deployed on thousands of commercial and recreational drones. The vast community of developers and users ensures refined controls code with lots of features.

Navigation

We have made slight modifications to the Arducopter firmware, which allow us to set the EKF origin. This has been done to allow our drone to navigate in GPS denied environments. For general waypoint navigation, the drone utilizes the Arducopter EKF that takes in data from the IMU, compass, altitude LIDAR and the optical flow sensor. The EKF position is then queried via mavros, so that our strategy code can use our position to make intelligent decisions. In addition, the strategy node takes in data from computer vision and 2d lidar. The strategy will determine a waypoint and mode, and the ros controls node will do a sanity check on the waypoint, then publish the waypoint until the drone has reached its destination. Over time, the EKF will have error in comparison to reality. To combat this, the control node will input a correction vector. This correction vector is formed from our computer vision algorithm that recognizes the absolute corners of the grid. This algorithm goes as follows. Gaussian blur, color threshold, erosion, dilation, Hough, line intersection, and non

maximum suppression.



Overview of control system

Flight Termination System

Our drone houses a 40 amp brushless esc connected to a rc receiver, which acts as our kill switch. In the event of catastrophic failure the safety operator can send the kill signal, and all power will be cut to the drones systems.

Payload

The Drone has been designed to carry all the necessary GNC sensors, 2d LIDAR, and array of cameras and 3 nvidia TX2 Jetsons. In addition the drone carries a telemetry radio, kill switch and rc receiver in order to carry out safe flight.

Sensor Suite

GNC Sensors

The quadcopter uses eight sensors for flight control. The main sensors used for the navigation of the quadcopter are a downwards-facing 1-D LiDAR module, a PX4FLOW optical sensor, and the built-in sensors of our flight control board, the Pixhawk 2. The 1-D LiDAR module used is the LiDAR-lite v3 from Garmin. This one dimensional sensor is directed downwards and provides data that is used to determine altitude. The PX4FLOW sensor measures velocity by comparing frame by frame images and measuring distance travelled and direction. The Pixhawk 2 contains three 3-axis accelerometers, three 3-axis gyroscopes, and two 3-axis compasses. The data from these sensors is used by the Pixhawk onboard system for flight control and the maneuvering of the quadcopter.

Mission Sensors

For the goal of completing Mission 7 the drone must be able to detect roombas and obstacles. The quadcopter is equipped with a single downward facing camera, four outward facing cameras, and a two-dimensional plane LiDAR sensor for obstacle avoidance. The singular bottom camera is a Logitech C210 webcam. This webcam is used for roomba detection and tracking the targeted roomba for autonomous interaction.For the four peripheral cameras are Logitech 960 C270. These are used to recognize and track the roombas around the playing field. These five webcams provides enough information allowing the drone to incorporate the recognized roombas into the strategy of the autonomous system. The 2D LiDAR sensor is the SWEEP sensor by Scanse.This is a single lidar that rotates to provide a full x-y plane

view inline with the z position of the drone for obstacle detection. We chose to use this for obstacle detection rather than four separate LiDARs or webcams to save on computing power.

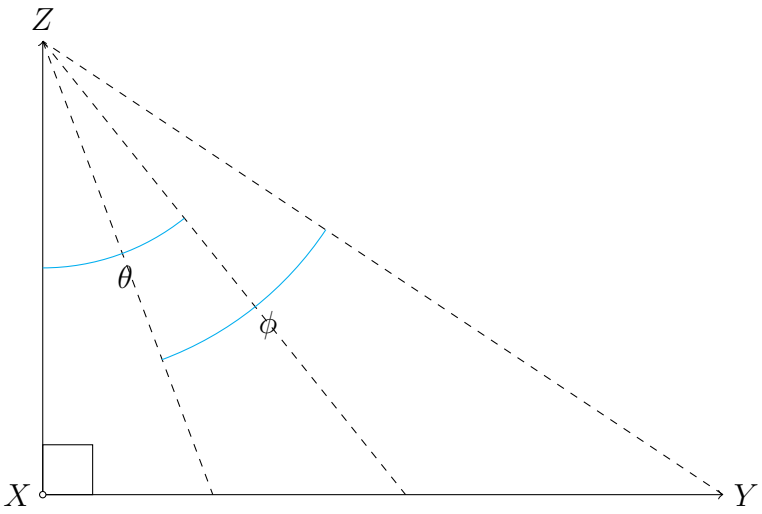
Target Identification

For target identification, the drone carries five Logitech C270 USB webcams. One of the cameras is located at the bottom of the drone and faces straight downward. The other four cameras are mounted on the bottom side of each arm facing outward. All five cameras are mounted for identifying as many target Roombas surrounding the drone as possible in order to send their coordinates to the strategy node.

There are two main objectives that need to be completed to successfully relay target information to the strategy node. The first objective is to take in the raw video data and detect the target Roombas so that pixel numbers of the centroids of the Roombas in the frame can be sent as output. The second objective is to take the pixel numbers of the centroids of the detected Roombas as input, and then output x and y coordinates relative to the drone using a coordinate transformation and the known height and orientation of the camera with respect to the floor.

In order to accomplish the first objective, an open source software package developed by Joseph Redmon called YOLO (You Only Look Once) is used. More specifically, the TinyYolo version of YOLO was elected to be used since it is lighter and can obtain higher frame rates. YOLO is a neural network that can identify user-defined objects once it has been trained. After training the neural network by feeding in approximate 2000 photos of the target Roombas (with the Roombas identified manually), YOLO can reliably identify target Roombas in real time. An example is displayed below:

After YOLO outputs the pixel locations of each identified target Roomba, the final objective is to transform these pixel coordinates into x-y coordinates relative to the drone. In addition to the pixel coordinate (PIX #), the other knowns are height (z), camera rotation angle relative to downward facing (θ), and field of view (ϕ), as well as total pixels in x and y (PIX_MAX). Here is a diagram to illustrate these knowns below:



In order to transform this pixel coordinate into x-y coordinates relative to the drone, a few key assumptions made are that:

- The camera behaves as a pinhole model
- Each pixel in the frame takes up an approximately equal fraction of the field of view

A big pitfall in this model is that no distortion from the camera lens is accounted for; these effects are still being investigated.

Transformation:

Pixel given from YOLO: PIX Pixel number of the center:

$$PIX = \frac{PIX\_MAX}{2}$$

Define ϕ as portion of angle from center image to pixel detected:

$$\phi = z * abs(\frac{PIX}{PIX\_MAX} * \phi - \frac{\phi}{2})$$

To find the real coordinate ($REAL$) relative to the drone in the X direction, There are two cases:

If PIX lies above the center pixel, that is $PIX > \frac{PIX\_MAX}{2}$, Then:

$$REAL = z * tan(\theta + \frac{\phi}{2})$$

If PIX lies below the center pixel, that is $PIX < \frac{PIX\_MAX}{2}$, Then:

$$REAL = z * tan(\theta - \frac{\phi}{2})$$

There are two cases to finding the real coordinate ($REAL$) relative to the drone in the Y direction:

If PIX lies above the center pixel, that is $PIX > \frac{PIX\_MAX}{2}$

$$REAL = z * tan(\theta - \frac{\phi}{2})$$

If PIX lies below the center pixel, that is $PIX < \frac{PIX\_MAX}{2}$

$$REAL = z * tan(\theta + \frac{\phi}{2})$$

This is different from the x only because top of frame is 1 and bottom is MAX_PIX, while in X 'top' of frame is MAX_PIX and bottom is 1.

Once the coordinate X,Y in meters is found relative to the drone, it must be transformed into a coordinate that is in the gym reference frame. The gym reference frame is defined as a right handed coordinate system starting in the bottom left hand corner of the field where the green line meets the outer white line. This transformation is defined as followed.

$$yawAngle = currentHeading - GymOffset$$

$$roombaPosGym = CR + EKFdronePose + R(3)(YawAngle) * roombaPosDrone$$

$$R(3) = \begin{bmatrix} cos(yawAngle) & -sin(yawAngle) & 0 \\ sin(yawAngle) & cos(yawAngle) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

CR is the correction vector formulated from the corner detection vision algorithm. EKF-dronePose is the position of the drone queried from the pixhawk. R(3) is the rotation matrix about the third axis. GymOffset is the direction of the 2 axis defined during the startup sequence before flight. The new coordinate in the gym frame is passed off to the strategy node as data to be acted on.

Communications

Power Management System

The quadcopter uses a 10000mAh 4S 10C LiPo battery and a 2000mAh 3S 15C LiPo battery for power. The 10000mAh 4S 10C LiPo is used to power our ESCs and Motors. With 4 T-Motor U3 K700 motors running at close to the ideal 50% throttle, a 4 Cell battery was necessary to reach the 1800 grams of thrust per motor needed for hovering flight. The 2000mAh 3S 15C LiPo will power the flight control board and Jetson compute units.

$$T(min) = Batterypower(AH) * 60/Totalcurrentdrawnbymotors$$

Assuming an average throttle of 65% for maneuvering, we see a 9 A current draw from each motor, and with a 10 Ah battery, that provides us with 16 minutes of flight time with minimal maneuvering and height change. This gives us plenty of headroom for more intense altitude and positioning changes, which will allow us to move more quickly during the competition, a valuable asset with the given time limitation.

$$MaxcontinuousAmpdraw(A) = Batterycapacity(Ah) * Dischargerate(C)$$

To save on weight and space, we chose a 10C discharge rate battery, as this is still more than enough to match the maximum continuous current draw of 25 A and maximum practical current draw of 19.4 A required by the motors. The 10000 mAh battery with a 10C discharge rate gives us a maximum continuous current draw of 100 A, matching the limit of the motors.

The eCalc online tool was used to validate our calculations, and as seen by the generated graph below, there should be no issue reaching above-10 minute flight times at speeds below 15 mph

INSERT GRAPH

As for the 2000 mAh 3S battery, this battery powers our Pixhawk flight controller and our three Jetson TX2 compute units. This was done partly to dedicate the entirety of the main battery to propulsion to maximize flight time. The other reason is that the speed controllers we chose (T-motor Air 40A) have no BEC, so a separate battery for the flight controller (and in turn the receiver) was the simplest solution. 2000 mAh was chosen because of the slim form factor we were able to find it in, and because it will provide more than adequate power while not adding any significant amount to the quad weight.

Operations

Flight Preparations

Checklist

1. Check battery voltage (12.0-12.6v) and insert
2. Make sure LIDAR and PX4FLOW are not covered
3. Make sure that ESCs are plugged in to the right pins
4. Make sure prop guards are secure

5. Make sure that the props are on in the correct direction (leading edge in)
6. Make sure that the props are not upside down
7. Make sure props are clear of any structures
8. Make sure Kill Switch is in off position
9. Power on controllers
10. Make sure TELEM2 is unplugged
11. Plug in power loop
12. Turn Kill Switch to on position
13. Make sure ground station has good telemetry
14. Verify critical sensors are giving good data
15. Plug in TELEM2
16. SSH start/verify autonomous scripts are running
17. Make sure everyone is clear of drone
18. Verify mode switch in Stabilize
19. Hold safety button
20. Mode switch to Autonomous

Man/Machine Interface

The drone has a DX7 RC controller interface for system checks and emergency situations. In addition, the flight controller can be monitored via a telemetry radio using Mission Planner. Our autonomous scripts must be initiated by an SSH connection to our Nvidia Jetsons.

Risk Reduction

Vehicle Status

Shock/Vibration Isolation

The modern pixhawk flight controller contains internal damping from the vibration of the drones structure. The decided there is no need for extra damping. After completion of our first drone last year, flight logs from the Pixhawk confirmed that vibration on the built quadcopter was well within recommended amounts. The team believes there is no need to have landing shocks, because the sophisticated Arducopter flight stack has logic to recognise landing. Because of this, nominal landings will not interfere with the guidance or navigation of the drone.

EMI/RFI Solutions

The Arducopter flight stack helps keep interference risk down through the EMI calibration on the Pixhawk. The calibration calculates the magnetic interference correction by formulating a function of the magnetic interference based on the power setting of the motors.

Safety

We took significant steps to ensure safe operation of the quadcopter. In order to keep the quadcopter from flying where we do not want, we have a Spektrum receiver module onboard so our designated pilot can manually control the vehicle. Additionally, we are able to check the status of the quadcopter through our telemetry to our ground station. The structure of the drone is also outfitted with prop guards to mitigate damage to people or property in the event of catastrophic failure. Our prop guards took multiple iterations. We wanted a design that would not interfere with the prop wash, but still be able prevent damage to the quadcopter itself in the unfortunate event of collision. The first iteration looked nice, but interfered with the prop wash too much and weighed too much, reducing the thrust. However, the next iteration cut weight and did not interfere with the prop wash.

Modeling and Simulation

To safely test our software throughout the development process, the team implemented a simulation using the Gazebo application. Using this tool, TAR was able to deploy full scale software in the loop simulation without jeopardizing hardware with each design iteration.

To employ the Gazebo framework, the team imported a standard quadcopter model from the arducopter standard library and roomba models from Gazebo model library, which were then updated with the color coded plates and obstacle tubes prescribed for the competition. Additionally, the ground plain within the reproduction was changed to match the texture expected at the competition.

Gazebo was chosen because of its compatibility with ROS and Ardupilot, the internal communication protocol employed onboard our drone. With this congruence, the exact same software run onboard the drone could be run within the simulation, complete with sensor feedback detected within the simulated instance. C++ scripts could then be utilized to enforce the correct roomba behaviors, scripting the robots using the same messaging environment as the quadcopter itself (ROS).

Overall, modeling our software behavior within a Gazebo simulation greatly expedited the development process. By standardizing the simulation setup, tests could be conducted on multiple computers at any time, instead of requiring sluggish process of updating, calibrating, and flying the physical drone. Additionally, simulation protected the hardware from the accidents and unintentional flight behavior inherent to prototype software.

Testing

Simulation, however, can only prove so much. Eventually, refined software was tested on a model testbed or on the drone itself. Much of the computer vision and general software integration was developed on this testbed, where performance could be verified with the same sensors and computational hardware as the aircraft before full scale deployment. Even strategy nodes could be tested on this platform, with waypoint predictions vetted before being deployed to the aircraft.

Flight tests, expectedly, require a specific setting. Generally, Texas Aerial Robotics tested on the roof of the Aerospace building on campus at the The University of Texas. This controlled environment bore witness to hardware implementation throughout our design process, from original flight readiness to early autonomy.

However, summer development saw additional test flights conducted in a vacant parking garage, but both locations suffered from magnetic inconsistencies which interfered with compass calibrations. Because of this, later tests, including the qualifying run submitted, were conducted in a parking lot to the north of the university.

Regardless of the setting, each flight test was conducted with constant apparatus. A four meter by four meter test mat printed with the pattern expected at competition, to allow optical flow for navigation. Across this mat drives a test roomba, which accurately recreates both the physical appearance and behaviors of those robots that will be tracked at the venue, complete with top plate and switch. Finally, there is the drone itself, which starts on a corner of the mat, takes off autonomously, and then executes the maneuver to be tested.

Conclusion