Intelligent Quadcopter for IARC 2018

Umer Salman, Mario Gonzalez
The University of Texas at Austin

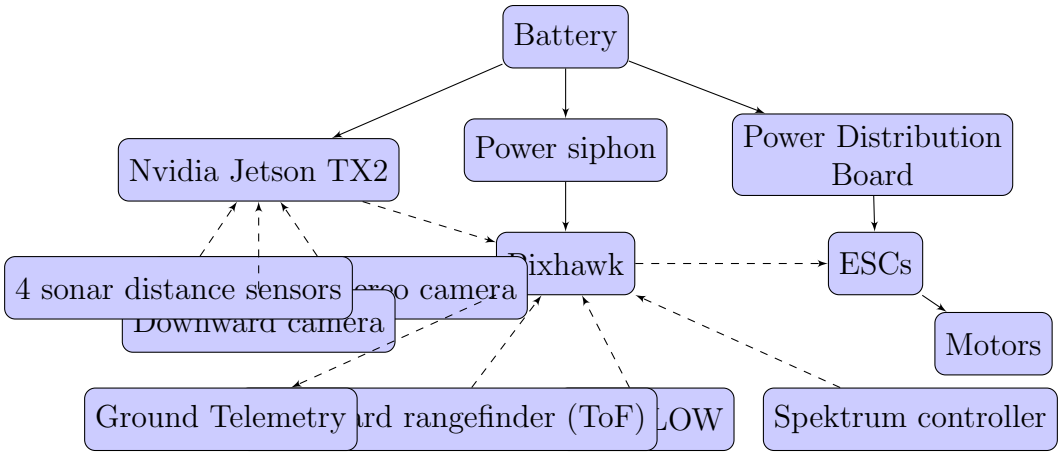
ABSTRACT

Texas Aerial Robotics has constructed a custom quadrotor aircraft for the fulfillment of the International Aerial Robotics Competition Mission 8. A combination of lightweight materials and powerful propulsion optimizes flight time, while standardized flight control systems supported by a massive community enables customization and adaptation to the mission at hand. This includes the integration of a multitude of sensors that enhance environmental awareness and allow for the aiding of people to analyze and interact with an environment. This is done through homebrewed QR code decoding algorithms and the shared computing power onboard the multiple aircrafts. Safe development has been assured by strategic selection of components, thorough vetting of software in Software in the Loop simulation, and robust testing procedures in secure areas.

INTRODUCTION

Previous aerial robotics missions have always allowed for GPS or set navigation points that allowed for constant locational awareness for the drone. Location data is important to any mission, as telemetry prevents various failures, including complete loss of control for the drone. Thus, Mission 8 requires novel solutions to deriving this critical telemetry as GPS is forbidden. Additionally, the aircrafts must be capable of taking commands in a humanized "natural" fashion and avoid other aircrafts without being informed of those other aircrafts' locations.

Texas Aerial Robotics addresses these challenges by utilizing computer vision. TAR uses ORBSLAM2, an "Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras" to create maps of the arena while in flight and use that data in real time. The drone creates pointclouds of the objects in view using the stereo camera and combines sections together. This means that should a section of the map start showing drift, the ORB-SLAM2/ORBSLAMM System will toss out just the section while maintaining as much of the map as possible. TAR hopes to combine the processing between the multiple drones to have them work together on one map. These drones also solve for the QR code that is the combination the human needs for collecting the objects. The drone is also built with safety for a plethora of other objects, containing man-safe propellers for any humans and sonar sensors for avoiding other air vehicles at all times.



Flowchart of quadcopter electronics
**Solid arrows indicate power while dashed lines indicate power+data*

Yearly Milestones

2019 - Have a working system, with localization of multiple drones working together to solve the QR code and communicate with the person through voice commands in working or nearly working state.

2020 - Finish all aspects of mission through more networking between the quadcopters, tweak the physique of the drone for better flight control and stability, have better communication between drones to know where enemy drones are at all times, add aesthetic design choices, and test for hours in gyms in as many conditions as possible,

AIR VEHICLE

Propulsion and Lift System

The vehicle utilizes an y axis symmetrical quad-rotor system. This H-frame provides a stable quadcopter for a smaller overall package while maintaining plenty of room for electronics. The frame is lifted by four T-Motor MN2212 KV920-V2.0 motors. These motors provide 1.8 [kg] of thrust per motor at 100% throttle when paired with a 4-cell battery and 9.5 [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 9.5 [in] long and made of plastic. 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

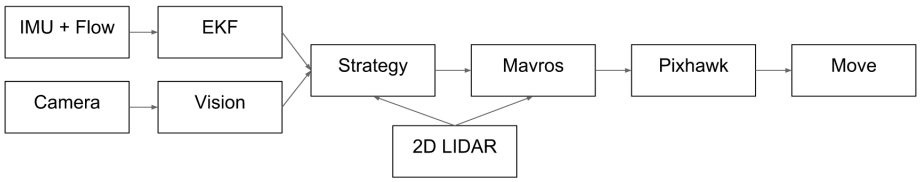
Stability Augmentation System

Our system builds off of the robust open source project ArduCopter. ArduCopter, which runs on the Pixhawk, controls the stability of the drone by taking in data from the IMU, compass,

altitude ToF, 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 make our drone capable of navigating in GPS denied environments. For general waypoint navigation, the drone utilizes the ArduCopter EKF that takes in data from the IMU, compass, altitude ToF, and the optical flow sensor. The EKF position is then queried via mavros so that our controls code can use our position to make intelligent decisions. In addition, the controls node takes in data from computer vision and the sonar sensors. The controls node determines a waypoint and mode, then does a sanity check on the waypoint before publishing the waypoint until the drone has reached its destination. Over time, the EKF will have some error in comparison to reality. However, using ORBSLAMM, an open-source version of ORBSLAM that contains multimapping, we are able to combat this by correcting the maps through physical objects (which will not be moving), which is then used to correct the EKF.



Overview of control system

Flight Termination System

We will be using ArduCopter's EmergencyStop to act as our killswitch. In the event of catastrophic failure, the safety operator can send the kill signal, and all power will immediately be cut from the drone's ESCs, and in turn, the motors.

PAYLOAD

The drone has been designed to carry all the necessary GNC sensors, an array of sonar sensors, a stereo camera, a downward camera, and a Nvidia Jetson TX2. In addition, the drone carries a telemetry radio, WiFi antennas, and an 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 include a downward-facing 1D LiDAR module, a PX4FLOW optical sensor,

and the built-in sensors of the Pixhawk 2 flight control board.

The 1D LiDAR module used is a Time of Flight (ToF) sensor named the Teraranger One. This one-dimensional sensor is directed downward and provides data to determine altitude. The PX4FLOW sensor measures velocity by comparing frame by frame images and measuring the distance traveled 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 maneuvering of the quadcopter.

Mission Sensors

For completion of Mission 8, the drone must be able to detect the QR code and avoid obstacles, both moving and non-moving. The quadcopter is equipped with a single downward facing camera for reading the quadrant of the QR code. A stereo camera is used for ORBSLAMM for localization. There is also an array of sonar sensors for obstacle avoidance.

The singular bottom camera is a ELP webcam. This webcam is used for reading the QR code quadrant. The stereo camera in front is also an ELP camera. These cameras were chosen for their very light weight while still allowing adjustment of exposure and other parameters to better read a QR code on an iPad. The array of sonar sensors contains 4 MAXBOTICS 7m I2C sonars. These allow the drone to detect objects on the sides the quadcopter intends to move in. We chose to use this for obstacle detection rather than a 2D LiDAR or webcams to save on weight, space, and computing power. Sonar sensors are very tried and true. Additionally, we opted for sonar rather than radar as radar has a harder time picking up smaller objects, like a drone.

Target Identification

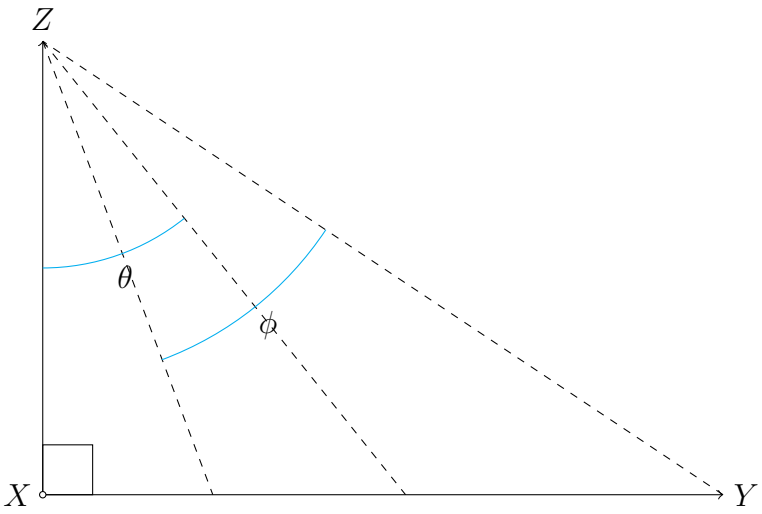
For target identification, the drone carries four Logitech C270 USB webcams and one Logitech C210 USB webcam. One of the cameras is located at the bottom of the drone and faces straight downward. The other four cameras are mounted on the bottom side of each arm facing outward. All five cameras are mounted 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 framerates. 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: $PIX\_CTR = \frac{PIX\_MAX}{2}$

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

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

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 $PIX > PIX\_CTR$, Then:

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

If PIX lies below the center pixel, that is $PIX < PIX\_CTR$, Then:

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

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 $PIX > PIX\_CTR$

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

If PIX lies below the center pixel, that is $PIX < PIX\_CTR$

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

This is different from the x only because top of frame is 1 and bottom is PIX_MAX, while in X 'top' of frame is PIX_MAX 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 follows:

$$yawAngle = currentHeading - GymOffset$$

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

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

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 axes 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.

Threat Avoidance

Our avoidance system takes data from a 2D LiDAR that is used to identify the obstacle Roombas. This data is passed and the strategy node uses this data to weigh possible movements negatively based on their relative position to the obstacle Roombas. The controls node also will use the data, such as in the event an obstacle robot is within 1.5 [m], in which case controls node will issue a waypoint opposite of the minimum vector.

Communications

On the Jetsons, data is passed between software nodes via ROS. Data in between our multiple Jetsons is passed via ROS messages through direct ethernet connections. Commands to the flight computer are passed via serial connection with data encoded in MAVLINK messages.

Power Management System

The quadcopter uses a 10000 [mAh] 4S 10C LiPo battery and a 2000 [mAh] 3S 15C LiPo battery for power. The 10000 [mAh] 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 2000 [mAh] 3S 15C LiPo will power the flight control board and Jetson compute units.

$$T_{min} = \frac{\text{Battery power}(AH) * 60}{\text{Total current drawn by motors}}$$

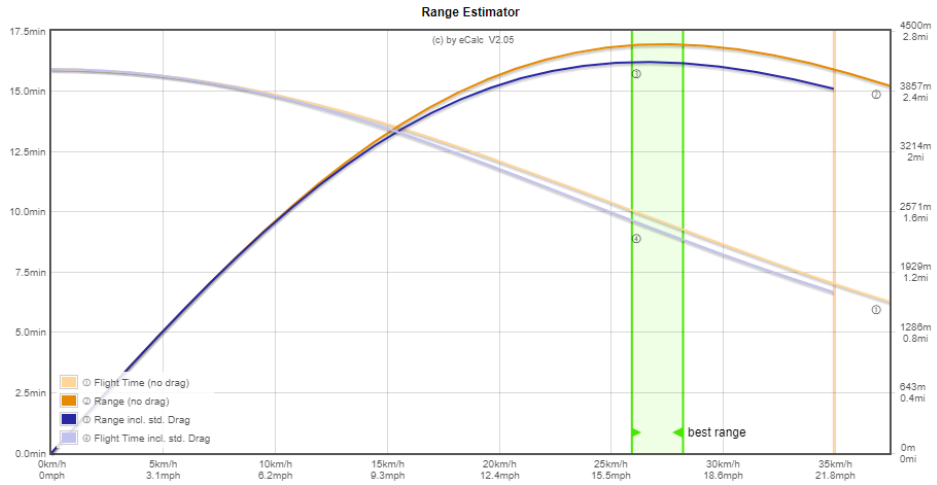
Assuming an average throttle of 65% for maneuvering, we see a 9 A current draw from each motor, and with a 10 Ah battery, that provides us with 16 minutes of flight time with minimal maneuvering and height change. This gives us plenty of headroom for more intense altitude and positioning changes, which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

$$\text{Max continuous Amp draw } (A) = \text{Battery capacity } (Ah) * \text{Discharge rate } (C)$$

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].



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 drone's structure. The team decided that 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, as the sophisticated Arducopter flight stack has logic to recognize 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 to prevent damage to the quadcopter itself in the unfortunate event of a 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 plane 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 the 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 University of Texas at Austin. 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

Texas Aerial Robotics' solutions to the challenges provided by IARC Mission 7 are bountiful and diverse. The physical design process, from the ground up, optimizes flight time while enabling the physical contact required for success. Unique software utilizes input from an expansive collection of sensors to overcome the telemetry limitations imparted by the contest and track the dynamic objectives as they move about the area. Safe operation of these vast networks of interlocking components has been ensured through particular hardware selection and thorough vetting.

References

[1] Autonomous Systems for S.P.A.C.E. R.O.B.O.T.I.C.S. Lab. general_info wiki, 2016. https://github.com/AS4SR/general_info/wiki/%23%23-Home-%23%23.

[2] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.

[3] Anne Solberg. Hough transform, 2009. https://www.uio.no/studier/emner/matnat/ifi/INF4300/h09/undervisningsmateriale/hough09.pdf.

[4] Ruye Wang. Canny edge detection, 2013. http://fourier.eng.hmc.edu/e161/lectures/canny/node1.html.