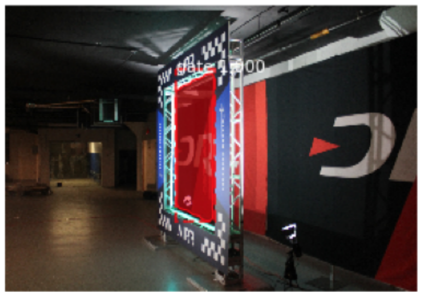
Technical Report: Test 2

SkyNet

1. Methods

Since a delay in perception propagates heavily onto subsequent models such as localization, path planning, and controls, we decided early on to focus more on inference speed rather than accuracy of intersection of union (IoU) scores. As long as sufficiently accurate locations (~IoU 0.5) for the gates can be correctly identified, reasonably large flyable regions could be extracted from the scene at any time step. From here, viable path planning waypoints can be formed that go through the center of the detected flyable regions. Hence, after analyzing multiple models such as YOLO variants [1], SSDs [2], and R-CNN variants [3], we chose Mask R-CNN [4] based on increased accuracy as well as the superior ability compared to previous versions in detecting smaller objects. This becomes useful when detecting gates at a greater distance. Mask R-CNN is one variant of Fast R-CNN, the key difference being an additional branch for predicting an object mask in parallel with an existing branch for bounding box recognition. Typically these R-CNN approaches are slow unless properly optimized (i.e. via TensorRT) and parallelized well, but our implementation is fast enough to account for this.

Our implementation is based on a Mask R-CNN general framework for object instance segmentation. The COCO dataset was initially used, and transfer learning was then employed on the set of pre-trained weights in order to more accurately perform the instance segmentation. Figure 1 shows a bounding box placed successfully over the flyable region of a gate within an image. As shown, the instance segmentation of Mask R-CNN outputs pixel level segmentation. This non-rectangular segmented data can be utilized to easily increase the IoU accuracy by at least a factor of 10% or more. Unfortunately we were unable to complete this within the deadline, but we plan to do it in our future works.



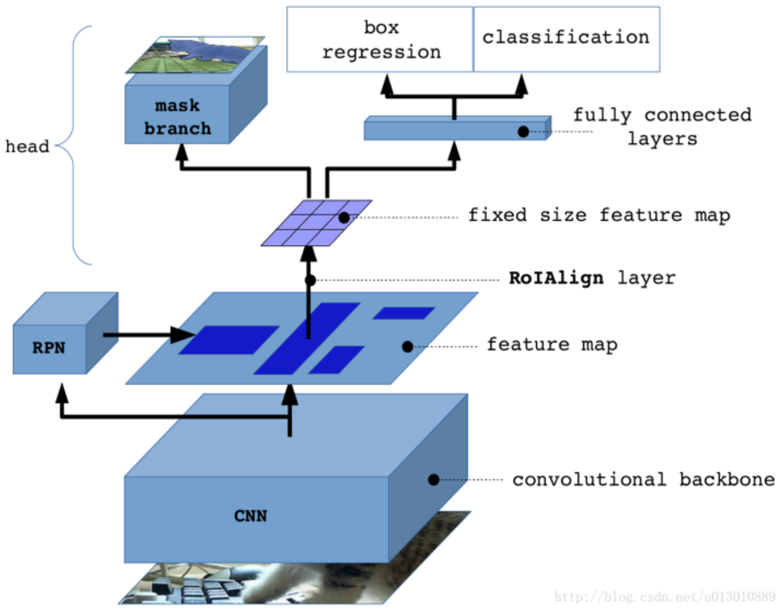


Figure 1: (left) A red bounding box around the flyable region of a gate, seen at a sharp angle. (right) a diagram of the Mask R-CNN architecture used in this approach ([4])

Our network was trained on two classes (gate and background). The images for training and testing were resized to 832x832x3 and we used Resnet50 as the backbone with anchors at 8, 16, 32, 64, and 128. We used a region of interest (ROI) of size 32 per image during training.

2. Future Works

Our team has several proposals for improving our results for more accurate gate detections and their application to real-life drone racing. To start off, data augmentation and analysis of data distribution will be important for further improve the accuracy of our approach. For this task all gates had one same real shape. However in a real drone race, the gates could have different shapes, sizes, and colors. Hence proper data augmentation will be required in order for our model to generalize to many types of gates that may be present in a race.

Subsequent model optimization will be done using CUDA or Tensor-RT optimization, which will shave off critical milliseconds from our computation time per image frame and increase the frame rate we will need to race. Additional model architecture refinement may also improve latency and accuracy. There is a natural tradeoff between the model accuracy and the latency, which we will seek to exploit. We intend to find this ideal tradeoff point once detection and localization/control are combined in silico. Altering the size of the image or using a different backbone network that is heavier or lighter will be studied. The Mask R-CNN inference time for a larger image, as well as a deeper backbone network, will generate a more accurate position in general; however, a smaller image or shallower backbone will allow for much faster computation of the bounding region. We also will not limit ourselves to Mask R-CNN, but will investigate other models that we deem beneficial and give us the mix of accuracy and speed we desire.

We will also take advantage of the fact that our current implementation does instance pixel-wise segmentation. After dividing the detected region into four quadrants, we can deduce the corners of the polygon by finding the point with the most euclidean distance from the center of the instance detection. This will further improve our IoU by allowing for non-rectangular flyable regions. We can also account for situations where there are obstacles such as pillars interrupting the flyable region by collecting all region instances located within certain threshold distances and treating them as belonging to a common flyable region, then finding the four corners as mentioned above. This will increase our IoU accuracy significantly, which can even enable us to estimate the distance to each gate by utilizing the homography of the detected gates.

For detecting distances to each gate as well as pixels, they can be easily computed using semi-global mapping (SGM) on image regions found to be within the frustum of the flyable zone. Using 2D box location along with distance information, the accuracy of our localization routine will improve substantially. However, if SGM methods are prove too computationally expensive for this challenge (particularly on the target hardware platform, NVidia Jetson), monocular visual odometry and depth estimation is a an alternative we will seek. Stereo camera SGM could also be used to generate training datasets for depth estimation training. Furthermore, we are aware that a combination approach utilizing both stereo imaging and monocular depth estimations can be combined to gain further accuracy [5].

We also wish to explore a variety of architectures for multi-task networks to regress the distance of objects and their detections as this technique is known improve both the distance estimation and the object detection accuracy in similar applications.

3. References

[1] Redmon, Joseph et. al. *You Only Look Once: Unified, Real-Time Object Detection*. CVPR (2016).
[2] Liu, Wei; Anguelov, Dragomir; Erhan, Dumitru; Szegedy, Christian; Reed, Scott; Fu, Cheng-Yang; Berg, Alexander. *SSD: Single Shot MultiBox Detector*. ECCV (2016).
[3] Girschick, Ross. *Fast R-CNN*. ICVV (2015).
[4] He, Kaiming; Gkioxari, Georgia; Dollar, Piotr; Girshick, Ross. *Mask R-CNN*. ICCV (2017).
[5] Saxena, Ashutosh; Schulte, Jamie; Ng, Andrew. *Depth Estimation using Monocular and Stereo Cues*. IJCAI (2007).

Technical Report: Test 3

SkyNet

1. Methods

Our approach to Test 3 can be broadly broken into three main components: localization, path planning, and control. Our localization is designed to take the IR beacon locations, stereo camera inputs and the inertial measurement units (IMU) to determine the current location of the drone in the simulation relative to the position of the gates. Path planning seeks to determine the optimal path through multiple gates based on the localization output. Finally, our control algorithms translate the path planning output into a specific action plan, then iteratively executes on these.

For localization, we chose to make use of the visio2 library built in to ROS, which takes stereo camera image as input, tracks harris corners and computes depth information, and using the resultant visual odometry to return the position within the simulated warehouse in three dimensions. Since the visio2 library cannot always be relied upon to give the exact position of the drone (particularly in instances where not enough features can be tracked for accurate visual odometry), we intended to also apply approximate gate locations (as measurement update) based EKF, along with IMU based state estimations. However, unfortunately due to lack of time, our gate based distance estimation was not fully developed into a satisfactory performance level, and IMU could not be incorporated. So for the submission we solely relied on the viso2 package. See future works section for the ideas planned to be implemented. Nominal gate location based EKF would have been critical as it is the only source that is robust against drift over time (which are inevitable for the visual odometry and IMU based state estimations).

Path planning takes into account the positions of the following four gates as well as the coordinates of the ego-drone from the localization algorithm, then finds a three-dimensional spline that passes through them. The function then returns a series of “waypoints” which are intended to divide the extended flight path into smaller, more manageable chunks. For each of the waypoints, the 3d position and velocity vector are calculated. As soon as a “waypoint” has been passed by the drone, the path is re-calculated. In the event that fewer than four gates are left to completion, the path planner adds an interpolated gate position that generates an identical number of waypoints no matter how many gates remain to be passed. The velocity is derived as a linear relation to the approaching curvature of the line from one waypoint to the next. Here we define the curvature as a value bounded between 0 and 1, which is the ratio of the straight-line distance to the line integral when looking a short distance ahead. In other words, the straighter the approaching segment of the spline is, the faster the drone will proceed. The more curved the approaching segment is, the slower the drone will proceed. We found this to be a helpful setting so the drone does not crash while making close-cutting turns. Also the path planning provides 3 directional vectors that is tangent to the waypoint location on the spline, which then later controller uses to compute the yaw angle of the desired direction at the next waypoint.

Our process of drone control can be summed up in Figure 1. There are three separate controllers that are all working in concert to give directives to the drone. These include an altitude controller, a yaw controller, and a roll-pitch controller. The altitude controller takes the desired altitude and the z-direction velocity and position at a given time t as input from the localization algorithm and returns the desired thrust. The computed thrust incorporates the required force to counter gravity as well as the force to ascend. The yaw controller acts to return the angular rotation along the z-axis based on the desired output from the path planning. The roll-pitch controller takes the desired next waypoint, the current drone position and orientation and computes pitch rate and roll rate. The pitch rate is proportional to the desired pitch which is proportional to the optimal velocity computed by the planner. The roll, on the other hand, calculates the translational error from the current location and the planned path and corrects to that. The body-rate controller takes each of these values and controls the drone accordingly. Together with the thrust calculation from the altitude controller, we have all the control parameters necessary to fully control all axes of motion that the drone can utilize.

A number of difficulties are encountered when trying to optimally control the drone. Each of the controllers is a proportional-integral-derivative (PID) controller, and therefore must be tuned at all three levels simultaneously for proper control to be achieved. There are still a number of issues that can arise with the controllers being ill-tuned, which can cause the drone to over-correct and crash. We ran out of time to apply PID tuning such as twiddle and for more complicated scenarios, but we did what we could to tune them as best as possible.

While there are certainly a lot of improvements that should be made, we believe we have the architecture in place to reliably adjust and significantly enhance our performance in the near future. **However, going forward with non-simulated environment, using PIDs and tuning them would become prohibitive to suit all unknown environment noise conditions such as wind. See Future Work section below for details on how we plan to tackle them.**

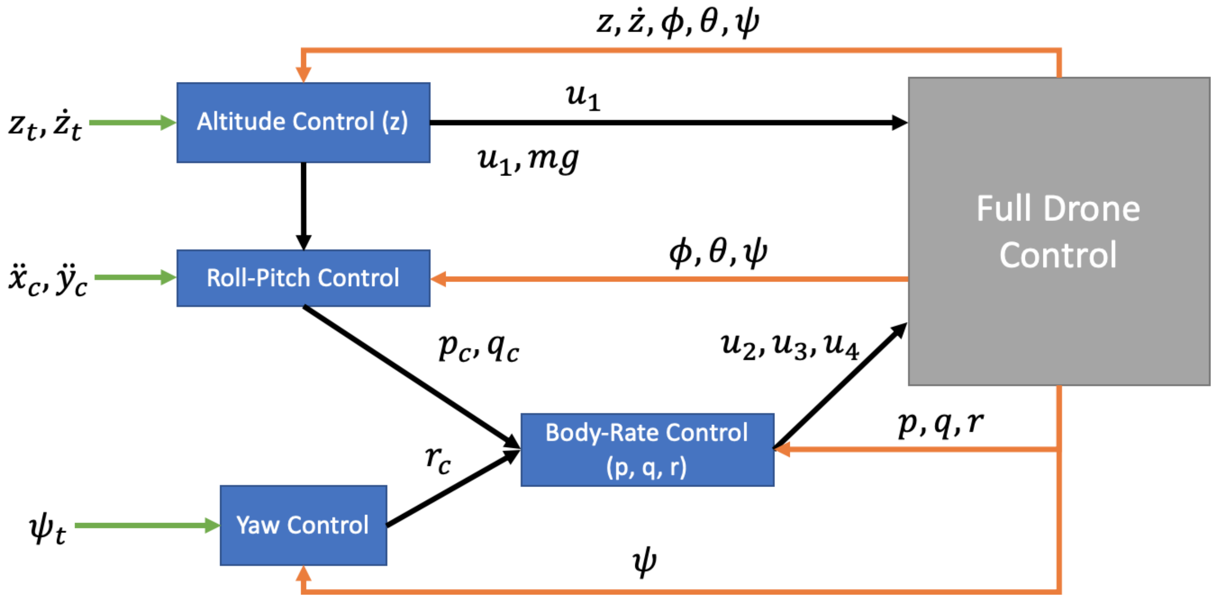


Figure 1: Drone control diagram

2. Future Work

For improved localization, we hope to also incorporate an advanced monocular/stereo visual odometry that takes into account pixelwise depth estimation as well which directly regresses the translational and orientational offsets per timestep. This combined with IMU based estimation and gate-location based estimation in the framework of EKF, we hope to achieve the ideal localization accuracy we need.

For path planning, we will either employ A* or similar path planning algorithm to go around obstacles in the scene effectively with the lowest cost path. For heuristic measures, we will incorporate the curvature information so that the least curved path can be selected.

Finally, for controls, we are looking to apply reinforcement learning. Since the action space is infinite, rather than discretizing the action space and using Q learning variants, we will seek policy gradient methods such as PPO. With such algorithm, the controls will be trained to be robust against external forces such as wind drafts and perturbations or the presence of other drones. We will use the simulation environment for maximum training, but will mix the memories gathered from real environments as much as possible to train the policy optimally.