GoD: Perception Based Drone Racing by Spleenlab.ai

Florian Ölsner[1], Stefan Milz[1] and Florian Ludewig[1]

***Abstract*— Perception based aerial autonomy is still an ill-posed task. The NeurIPS competition ”Game of drones” (GoD) wants to tackle this problem with the aid of a live competition. The following manuscripts describes team Spleenlabs approach for the perception based track within the competetion: ”Tier 2”. The architecture is based on a robust, redundant and modular pipeline using CNNs and policy algorithms. For training data generation we propose a specific data extraction procedure using Airsim to create heterogeneous data-sets for robust object detection as base for policy and planning tasks.**

I. SYSTEM ARCHITECTURE

Spleenlab's Racing Approach is fully oriented on the perception task (see3). Therefore, two CNNs process the current RGB frame coming from the simulation engine. A Faster RCNN[1] using a Resnet50 backbone[2] predicts all visible gates. The model is trained as a single class predictor. Additionally, a customized Resnet18[2] is used to predict the four edges of the next gate. Those information are in pixel-coordinates. A simple depth regressor is used to inject sparse pseudo depths to interrupt for collision avoidance. All estimation are piped into the policy module that optimises the next flight controlling step based on the perception. Two consecutive frames are used to achieve an approximate 3D depth based on the 2D objects. IMU and noisy gate poses are used to start with good guesses or the remove outliers. The policy creates a stack of gate beliefs (3D positions) that are used for the planning. The policy detects gate transitions and processes the stack.

A. Faster RCNN Object Detection

Our Resnet50 backboned FasterRCNN is trained 5000 images including only that are splitted in a 90:10 (see **Results** manner for evaluation and **Data Generation** for the dataset specification.) The model uses the state of the art ROI Pooler and a class detector such as Box Regressor.

B. Closest Gate Edge Detector

We found that very close gates were sometime unstable predicted by the FasterRCNN. Hence, as a redundant path, we injected a customized edge detector that is based on a Resnet18 backbone and eight regression values per equivalent to the u,v coordinates of the 8 edges.

[1]Florian, Stefan and Florian are with Spleenlab.ai who supported the work.
Hauptstraße 18, 07929 Saalburg-Ebersdorf, Germany
florian.oelsner@spleenlab.com
stefan.milz@spleenlab.com

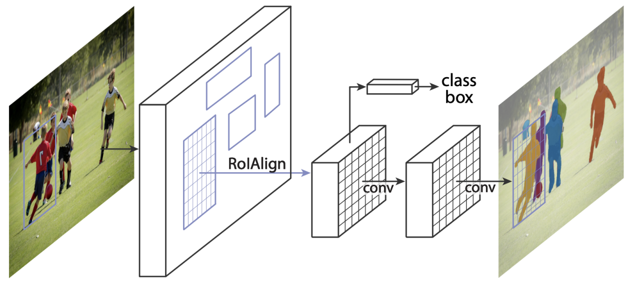

Fig. 1. FasterRCNN architecture for gate detection [1]

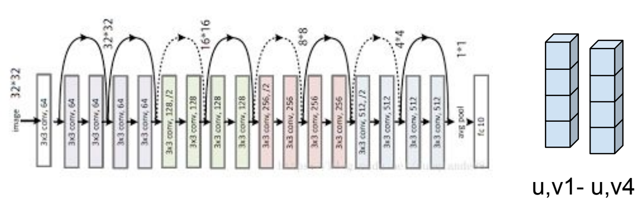

Fig. 2. Resnet18 architecture for closet gate edge detection [1]

C. Stereo Gate Matching

The Stereo Gate Matcher uses pixel coordinate based gate predictions from the current and the previous frame such as the IMU. We have to note the frame steps are not defined by the camera, but by the flying policy itself. Based on that, the 3D space projections $(t, t-1)$ are processed by a simple numeric optimizer, who finds the closest distance and creates an approximate depth for the detected gate. The gate optimizer can process several gates depending on the visibility of the RGB FoV. As a result a stack of next gate beliefs is created for the policy to handle a successful flight control. In a looping manner the gate beliefs will be updated by the matcher and the predictors.

D. Spleenlabs Racing Policy

Spleenlabs racing policy, which directly controls the flight controller and processes the results of the predictors can be explained with the following pseudo code example. The policy is optimized the succeed the perception track (see algorithm 1).

II. DATA GENERATION PIPELINE

Since, we focus on ”Tier 2” our data generation pipeline was designed to run on the ”Zhiang medium” environment within the training binaries of Airsim. However, due to their

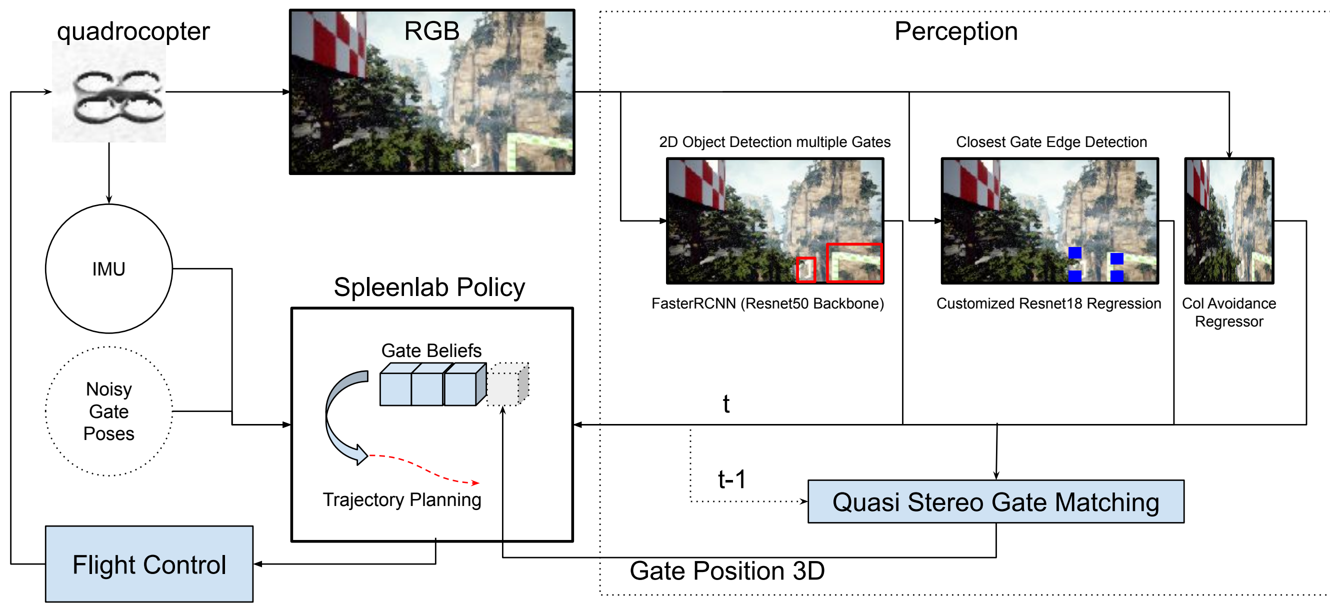

Fig. 3. Principal system architecture of the Spleenlab perception racing approach

Algorithm 1 Racing Policy

Require: noisyGatePositions
  gateBeliefs ← copy(noisyGatePositions)
  while not finished **do**
    if gatePass == TRUE **then**
      moveForwardForAShortDistance()
      currentGate++
      if |currentBelief.z - getImu().z| > threshold **then**
        flyForwardToZLevel(gateBeliefs[currentGate])
      end if
      turnTowards(gateBeliefs[currentGate])
    end if
    image ← getImage()
    gateBoxes ← predictGateBoundingBoxes(image)
    gateCenters ← computeConfGateCenters(gateBoxes)
    for each center in gateCenters **do**
      i ← associateCenterToGate(center, gateBeliefs)
      center3d ← getDepth(center, gateBeliefs[i].x)
      if close(center3d, gateBeliefs[i]) **then**
        gateBeliefs[i] ← mean(gateBeliefs[i], center_3d)
      end if
    end for
    velocity ← appropriateVel(gateBeliefs[currentGate])
    moveTowards(gateBeliefs[currentGate], velocity)
  end while

generic base it could be applied to all environments and switching levels could be used to increase diversity in future tasks. The if this method focus was to extract RGB drone images i_i with a size of n_s samples all drawn from realistic drone poses p_d with a wide variety in terms of position and viewing angles (see algorithm 2). Our datasets inlude RGB images $m_i \in R^{w \times h}$, instance image gate masks $m_i \in R^{w \times h}$, boxes per image $B_i \in (x, y, w, h) = \{B_{i1}...B_{it}\}$ and depth values per gate $d_i = \{d_{i1}...d_{it}\}$ and edges for the first gate $E_i \in (x1, y1, x2, y2, x3, y3, x4, y4)$. Poses of the drone are randomly drawn on the connection line of two random gates with a stochastic spawn range, a noise operator and an incidentally yaw rate. Exemplary, Fig. 4 shows a sample of a drawn drone pose in simulation mode with accompanied RGB image, instance mask and boxes.

Algorithm 2 Generate data for OD and Regression

Require: all true Gate poses G_{true}
  for $i = 1$ to n_s **do**
    p_d, gateID ← getRandomPose(G_{true}, configuration)
    drone.teleport(p_d)
    i_i, m_i, B_i, d_i, E_i ← capture(drone)
  end for

III. EXPERIMENTS AND RESULTS

A. Detection

For our experiments we have drawn 5000 images into our dataset. All images contain gates. We splitted into an train and eval subsets in a 90/10 manner.

- FasterRCNN: We run experiments on the dataset, where only boxes with a size of more than 5 pixels were exported. We used the SGD optimzer, an intial learning rate of $5.e-3$, an momentum of 0.9 and the combined FasterRCNN loss [1]. After 10 epochs we achieved the following results shown in Tab.II (Average Precision (AP), Average Recall (AR) investigating different sized boxes (all, small, medium, large) regarding the IoU

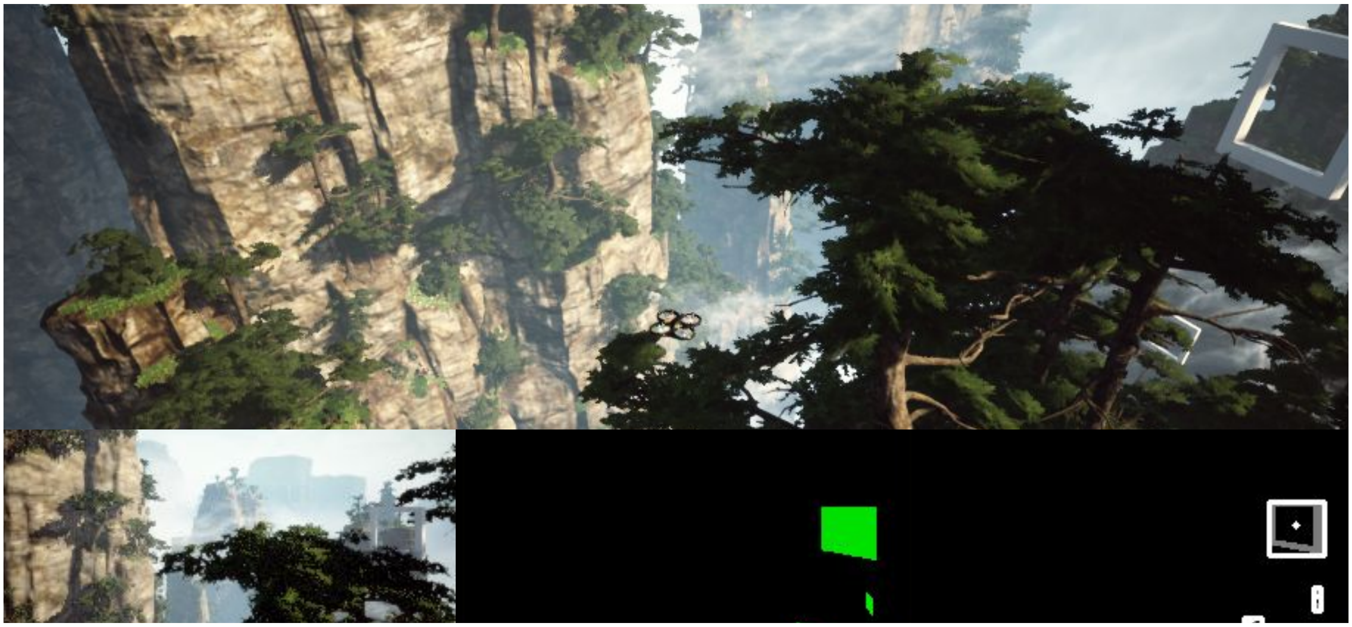

Fig. 4. Data Generation with Airsim: Learning to predict the Gates needs a randomly drawn drone pose with RGB images, instance masks and bounding boxes. Additionally we store 3D poses of the gates.

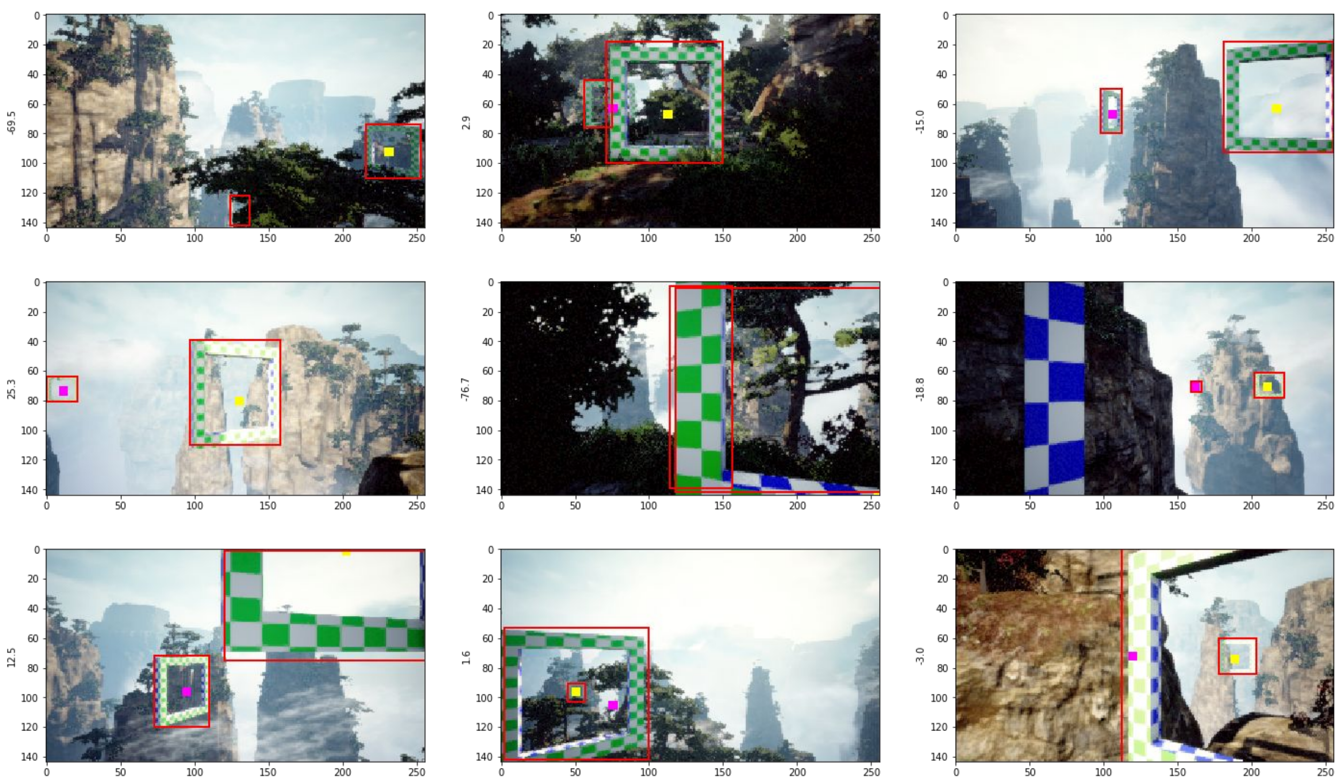

Fig. 5. Qualitative gate detections with FasterRCNN

(Intersection of Union)). Fig. 5 shows qualitative results on the eval dataset.

- Costum edge detections: Our costume edge detector runs in a simple regression manner on eight data points using a smooth L1 loss with an ADAM (initial learning rate l=1e-4) optimizer for 200 epochs. Qualitative results on the competition track can be seen in Fig.6

B. Drone Race

The results of our full pipeline are shown on the Leaderbord in Tab.I.

TABLE I
LEADERBOARD GOD TIER 2 (21TH OF NOVEMBER 2019)

Team	Num Gates Passed	Lap Time (s)	Num Gates Missed	Num Gates Attempted	Max Speed (m/s)	Avg Speed (m/s)
USRG	14	58.59	0	14 / 14	18.95	4.75
Spleenlab	14	76.28	0	14 / 14	18.77	4.31
Sangyun	14	80.70	0	14 / 14	8.65	3.71
Kukks	7	208.45	7	14 / 14	13.06	3.94

Fig. 6. Qualitative edge the detections of the closest gate

TABLE II
FASTERRCNN RESULTS AFTER 10 EPOCHS

(AP) IoU=0.50:0.95	area=all maxDets=100 = 0.926
(AP) IoU=0.50	area= all maxDets=100 = 0.997
(AP) IoU=0.75	area= all maxDets=100 = 0.970
(AP) IoU=0.50:0.95	area= small maxDets=100 = 0.922
(AP) IoU=0.50:0.95	area= medium maxDets=100 = 0.921
(AP) IoU=0.50:0.95	area= large maxDets=100 = 0.963
(AR) IoU=0.50:0.95	area= all maxDets= 1 = 0.823
(AR) IoU=0.50:0.95	area= all maxDets= 10 = 0.943
(AR) IoU=0.50:0.95	area= all maxDets=100 = 0.943
(AR) IoU=0.50:0.95	area= small maxDets=100 = 0.945
(AR) IoU=0.50:0.95	area= medium maxDets=100 = 0.934
(AR) IoU=0.50:0.95	area= large maxDets=100 = 0.970

IV. CONCLUSIONS

We have shown a remarkable approach for drone racing based on state of the art perception tasks.

ACKNOWLEDGMENT

We would like to thank Spleenlab.ai for support of this work.

REFERENCES

[1] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 91–99. [Online]. Available: http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385