Supplementary Material to
‘AirCapRL: Autonomous Aerial Human Motion Capture using Deep Reinforcement Learning’
IEEE RA-L and IROS 2020.

Rahul Tallamraju, Nitin Saini, Elia Bonetto, Michael Pabst, Yu Tang Liu, Michael J. Black and Aamir Ahmad *
†

1 Introduction
We first present a few additional discussions regarding our experiments. Further, we provide the details of all the deep neural networks (DNN) developed for the methods presented in the article. Algorithm and implementation details, network architecture and instructions for downloading and running the code are also provided.

2 Additional Discussion for Experiments: Reasoning for Network 1.1 losing the person often
When the only reward concerns centering (Network 1.1), there is only a single image point constraint for the MAV’s to keep. In this case, due to the underactuation of the MAV agents and the the fact that the camera is rigidly attached to the MAV frame, the MAVs tend to lose the person completely while making fast and aggressive maneuvers while maintaining the single point constraint.

On the other hand, when rewarded for only MoCap-related objectives (Networks 1.2 and 1.3), the agents become constrained by many more points on the image plane. Hence, they are more likely to keep the person anywhere on the image plane, irrespective of how far from the image center. Clearly, then combining both these rewards helps to achieve best CPE, while the MPE remains similar to the agents that only got MoCap objective-related rewards.

3 The AirCap Pipeline for ‘real observations’
The AirCap’s real robot pipeline involves

- a full fledged state estimation method for estimating person’s 3D position using camera images, running a convolutional neural network-based (CNN) detector and then a cooperative detection and tracking (CDT) algorithm on the CNN-based detections,
- the teammate positions obtained through wireless communication where every MAV determines its own position through a cooperative self-localization method using GPS, IMU, barometer and the communicated bias-corrections based on the cooperative estimation of the person, and
- MPC-based perception-driven active formation control.

In simulation, the AirCap pipeline executes all these, including the CNN-based detector, CDT, the cooperative self-localization method, communication packet loss, GPS noise and the MPC-based formation control. These are well described in detail in our previous works ([1] and [2]). Running the AirCap pipeline with any network policy in this work means replacing only the MPC-based formation control module with the network policy. All other sub-modules run as described above, thereby producing realistic observations.

* Authors are with the Max-Planck Institute for Intelligent Systems, Tübingen, Germany.
† {firstname.lastname} @ tuebingen.mpg.de
Algorithm 1 Pseudocode for centralized training

| Input: initial policy parameters \( \theta_0 \), initial value function parameters \( \mu_0 \) |
| for m in Total Episodes |
| • Collect trajectories \( D_m = \{s_t, a_t, r_t, s_{t+1}\}, t = 1 \ldots T_{episode} \) |
| • from parallel Gazebo runners, with 2 agents using policy \( \pi_{m,\theta} \), estimate advantage \( \hat{A}_{m,\theta}(s_t, a_t) \) using the value function \( V_{m-1,\mu}(s_t) \) and reward-to-go from \( D_m \). |
| • Maximize PPO surrogate loss \( L \) w.r.t \( \theta \) via SGD with Adam. |
| • Fit value function \( V_{m,\mu}(s_t) \) by regressing on current reward-to-go. |

4 Algorithmic Details

We train the agents using a centralized training and decentralized execution paradigm. Specifically, a centralized fully observable critic and an actor with local observations are learned by collecting experiences of all the robots simultaneously. The robots then execute this common shared policy to collect new data. The fully observable centralized critic and shared actor policy aid in maintaining a stationary environment for our two-agent problem. This enables us to use conventional single-agent reinforcement learning algorithms to train the multi-agent policy network. Algorithm 1 summarizes the training methodology used.

5 Network Architecture

Figures 3 and 4 in the main manuscript show our network training architectures. Both single and multi agent policies are two layer 256 \( \times \) 256 neural networks with ReLu activations. For estimating the advantage \( A_t \), the state-value network \( V_{\mu}(s_t) \) is approximated with a neural network with parameters \( \mu \). The value network architecture is a clone of the policy network at each stage. The current value function is fit by regressing over the reward-to-go estimate. We train the networks on Tensorflow with Adam optimizer and a stable baselines software implementation of PPO.

5.1 Additional details of Network 2.4

The architecture for this network is 256x256x128x64. It was first pretrained with behavior cloning against network 1.4 for 100,000 steps. The behavior cloning achieves a single agent following behavior for each agent in the environment. This network is subsequently trained using the centralized training and decentralized execution paradigm to train multiple agents using the MPE and CPE rewards. As mentioned in the manuscript this network is not penalized for collision avoidance in training. However, collision avoidance is applied as a potential field after the policy network generates a control action. Therefore the collision avoidance is part of the environment dynamics. In this case we use a stable baselines implementation: [https://stable-baselines.readthedocs.io/en/master/modules/gail.html](https://stable-baselines.readthedocs.io/en/master/modules/gail.html)

6 Code and instructions

Please download the code at [https://github.com/robot-perception-group/AirCap/tree/aircaprl/packages/simulation/my_firefly_training](https://github.com/robot-perception-group/AirCap/tree/aircaprl/packages/simulation/my_firefly_training) Please ensure that the branch ‘aircaprl’ is selected.

7 Training Setup

In this section, we elaborate on the experiments conducted for showcasing our proposed approach. We detail the hyperparameters used in training, the training setup, and RL environment definitions. Proximal policy optimization (PPO) was leveraged to train each network proposed in the submitted manuscript. The parameters for PPO are summarized in Table 1.

To enable parallelized training, we developed a networked reinforcement learning training setup over multiple computers. The computers communicate over the network using the ROS middleware in combination with a ROS-multi-master setup. Each Gazebo environment and its agents are associated with an independent ROS master. During neural network training,
| Parameter                        | Value        |
|---------------------------------|--------------|
| Discount Factor                 | 0.99         |
| Number of Steps per update per Env | 128         |
| Minibatches                     | 4            |
| Loss Entropy Coefficient        | 0.01         |
| Learning Rate                   | 0.00025      |
| Loss Value Function Coefficient  | 0.5          |
| Maximum Value for Gradient Clipping | 0.5       |
| Trade-off Factor for GAE        | 0.95         |
| Number of Epochs for Surrogate  | 4            |
| PPO Policy Clipping Range       | 0.2          |
| Optimizer                       | ADAM         |
| Activation Function             | ReLu         |

Table 1: PPO2 training parameters

Figure 1: Network 1.1 - Training Reward Curve. One contiguous training run in three parts. (Left) 0 – \(\approx 800,000\) steps. (Center) 800,000 – 3.2 million steps. (Right) 3.2 – 4 million steps

the transition and reward tuples from these environments are sampled over the network using a single computer which runs the PPO algorithm. Gazebo environments are modeled as OpenAI-Gym environments to ensure compatibility with the stable-baselines implementation of PPO\(^1\). We use a simulated human in Gazebo as the MoCap subject using the actor plugin\(^2\) and generate random trajectories using another custom plugin. We use four computers in total to train the neural networks. Since the rewards for reinforcement learning are modeled using multiple instances of state-of-the-art human pose and shape estimation neural networks, which are based on ResNet-50 architecture, the GPU requirements are high. The GPUs used during training include an Nvidia Titan RTX, Nvidia 1080 GTX, Two Nvidia Quadro P5000s. Different number of training environments are executed on each computer based on CPU and GPU usage. However, to test the networks, we only need a CPU as all the policy networks are shallow 256 \(\times\) 256 fully connected networks. For real robot experiments, we used a laptop running on an Intel Core i5-6300U CPU and 8 GB RAM. The neural network weights are updated after the algorithm samples 128 time steps of trajectories from each of the parallel environments. Therefore the batch size is 128 \(\times\) (Num of Envs). We use 4 minibatches per update. The ADAM optimizer with a learning rate of 0.00025 is used for optimizing the policy and value network losses.

8 Single-Agent Networks

In the following subsections we detail the parameters used in the single agent training environments and then showcase the reward curves obtained for each environment.

8.1 Network 1.1: Centering Reward

In the case of Network 1.1, the \(\psi_P^{f}\) does not include the person’s global-frame orientation component, i.e, this observation is invariant to how the person is oriented. The reasoning for this is that the centering performance would not depend on how the person is oriented w.r.t. to the robot. In Table\(^2\) the training environment parameters are described in detail. In

\(^{1}\)https://stable-baselines.readthedocs.io/en/master/modules/ppo2.html

\(^{2}\)http://gazebosim.org/tutorials?tut=actor&cat=build_robot
During testing, in simulation, the statistics (in Figure 5 of manuscript) indicate a first quartile value of 6, 520, 740, 380, 520 pixels and third quartile of 6, 740, 740, 380, 520 pixels for the quartiles. We observe that corresponding actor distance from the center of the image is 6, 520, 740, 380, 520, respectively for the quartiles.

Network 1.1 of the submitted manuscript, we utilize 8 Gazebo environments with a single MAV and single motion capture subject (or actor) in each. The actor’s velocity is sampled from a normal distribution with a mean of 1.4 ms\(^{-1}\) and a standard deviation of 1 ms\(^{-1}\). The actor’s trajectory is determined using a low-level go-to-goal controller which updates the actor’s heading goal by sampling a position from a uniform distribution. The actor’s velocity is sampled from a normal distribution with a mean of 1.4 ms\(^{-1}\) and a standard deviation of 1 ms\(^{-1}\). Inverting the reward shaping function provided in equation (4), we observe that corresponding actor distance from the center of the image is 100 pixels away. During testing, in simulation, the statistics (in Figure 5 of manuscript) indicate a first quartile value of 520 pixels from the center, median of 500 pixels and third quartile of 740 pixels. In real robot experiments, the statistics from the box plot indicate values of 280, 380, 520 pixels (in Figure 8 of manuscript), respectively for the quartiles.

### 8.2 Network 1.2 and 1.3: SPIN Reward and Weighted SPIN Reward

In the case of Networks 1.2 (and also 1.3 and 1.4), we feature-engineer some components of the observation \( \psi^P_t \) and provide it as an additional, redundant input \( \psi_t \) to the network. \( \psi_t \) is the bearing angle measurement of the person made by the robot. This feature engineering aids the neural network to understand that it would be rewarded only if the person is in the field-of-view. Network-1.2 is trained with environmental parameters detailed in Table 3. The training reward curve is as shown in Figure 2. We observe an average reward of \( \approx 240 \) for 1000 steps. This corresponds to a 3-D pose estimation error

| Environment Parameters     | Value                                                                 |
|-----------------------------|----------------------------------------------------------------------|
| Actor Velocity              | \( \|x^P\| \sim \mathcal{N}(1,4,1) \text{ ms}^{-1} \)                  |
| Actor Trajectory            | \( x^P \sim \mathcal{U}(-20, 20) \text{ m} \)                      |
|                            | \( y^P \sim \mathcal{U}(-20, 20) \text{ m} \)                      |
|                            | \( z^P = 1.1 \text{ m} \)                                          |
| MAV Position Limits         | \(-20 \leq x \leq 20 \text{ m}\)                                  |
|                            | \(-20 \leq y \leq 20 \text{ m}\)                                  |
|                            | \(5 \leq z \leq 8 \text{ m}\)                                    |
| MAV Velocity Limits         | \(-5 \leq vx \leq 5 \text{ ms}^{-1}\)                             |
|                            | \(-5 \leq vy \leq 5 \text{ ms}^{-1}\)                             |
|                            | \(-1 \leq vz \leq 1 \text{ ms}^{-1}\)                            |
|                            | \(-1 \leq \omega \leq 1 \text{ rads}^{-1}\)                      |
| Reset Environment           | 1000 time steps                                                     |
| Parallel Environments       | 8 environments                                                      |
| Total Training Steps        | \( \approx 4 \text{ million steps}\)                              |

Table 2: Network 1.1 - Environment Parameters
Table 4: Network 1.3 - Environment Parameters

| Environment Parameters  | Value                                      |
|------------------------|--------------------------------------------|
| Actor Velocity         | $\|\dot{X}_P\| \sim \mathcal{N}(1.4,1) \text{ ms}^{-1}$ |
| Actor Trajectory       | $x^p \sim \mathcal{U}(-20,20) \text{ m}$   |
|                        | $y^p \sim \mathcal{U}(-20,20) \text{ m}$   |
|                        | $z^p = 1.1 \text{ m}$                      |
| MAV Position Limits    | $-20 \leq x \leq 20 \text{ m}$            |
|                        | $-20 \leq y \leq 20 \text{ m}$            |
|                        | $5 \leq z \leq 8 \text{ m}$               |
| MAV Velocity Limits    | $-5 \leq vx \leq 5 \text{ ms}^{-1}$        |
|                        | $-5 \leq vy \leq 5 \text{ ms}^{-1}$        |
|                        | $-1 \leq vz \leq 1 \text{ ms}^{-1}$        |
|                        | $-1 \leq \omega \leq 1 \text{ rad s}^{-1}$|
| Reset Environment      | 1000 time steps                            |
| Parallel Environments  | 8 environments                             |
| Total Training Steps   | 2.4 million steps                          |

of 0.2 m averaged over all the joints of the actor. Similarly, Table 4 details the environment parameters for training the Network-1.3 with a weighted SPIN reward (equation (6) in manuscript). We observe an average reward of $\approx 200$ for 1000 steps, which corresponds to a weighted full-body average of 0.23 m for 3-D actor pose estimation error. Test time median errors for both the networks are $\approx 0.7 \text{ m}$ (as seen in Figure 5 of the manuscript).
### Table 5: Network 1.4 - Environment Parameters

| Environment Parameters       | Value                                      |
|------------------------------|--------------------------------------------|
| Actor Velocity              | $\|x_t\| \sim N(1.4, 1) \text{ ms}^{-1}$    |
| Actor Trajectory            | $x^T \sim U(-20, 20) \text{ m}$           |
|                              | $y^T \sim U(-20, 20) \text{ m}$           |
|                              | $z^P = 1.1 \text{ m}$                      |
| MAV Position Limits          | $-20 \leq x \leq 20 \text{ m}$            |
|                              | $-20 \leq y \leq 20 \text{ m}$            |
|                              | $5 \leq z \leq 8 \text{ m}$               |
| MAV Velocity Limits         | $-5 \leq vx \leq 5 \text{ ms}^{-1}$       |
|                              | $-5 \leq vy \leq 5 \text{ ms}^{-1}$       |
|                              | $-1 \leq vz \leq 1 \text{ ms}^{-1}$       |
|                              | $-1 \leq \omega \leq 1 \text{ rads}^{-1}$ |
| Reset Environment           | 1000 time steps                            |
| Parallel Environments       | 8 environments                             |
| Total Training Steps        | 3.5 million steps                          |

### Figure 4: Network 1.4 - Training Reward

8.3 Network 1.4: Centering and Weighted SPIN Reward

Table 5 and Figure 4 showcase the environmental parameters and reward curve, respectively, for Network 1.4. We observe that the average reward at convergence is $\approx 180$ for 1000 steps. The average per time step performance for both centering and SPIN rewards, assuming equal contribution from both rewards (both the rewards per time step are averaged during training), is $0.18$. This corresponds to $0.25 \text{ m}$ error in 3-D pose estimation and $125 \text{ pixels}$ error in centering the actor. At test time, we observe a median error of $0.7 \text{ m}$ and $250 \text{ pixels}$ in centering and 3-D pose estimation, respectively.

### 9 Multi-Agent Networks

9.1 Network 2.1: Centering, collision avoidance and AlphaPose Triangulation Reward

The parameters for the multi-agent training environment of Network-2.1 are as detailed in Table 6. Note that the actor’s root position is fixed for this network. However, the actor’s gait is animated, i.e., it emulates a walking motion without physically changing its position. The network is trained for 3 million time steps, and the reward curve is as shown in Figure 5. We note that the reward has a value of $\approx 0.8$ on average at convergence. This indicates good pose estimation, centering, and collision avoidance performance. At test time (Figure 6 in manuscript), we observe a median error of $\approx 200 \text{ pixels}$ for centering and $0.25 \text{ m}$ error in 3-D pose estimation.

In Fig. 6 we observe that the initial few steps of training incurs a negative reward. This can be attributed to high number of collisions at the start of training. It can also be observed that over the course of training, the agents learn to...
avoid each other so as to incur no negative rewards. Additionally, it is important to note that the agents are initialized to random positions in the vicinity of the person at each environment reset. Therefore, there are few episodes where the agent collide and crash at the beginning of an environment reset and might never recover until the next environment reset (1000 steps later). This could explain the abrupt valleys in the reward curve over the course of training. Similar reasoning is true for the next two networks.

### 9.2 Network 2.2: Centering, collision avoidance and Multiview HMR Reward

The parameters for the multi-agent training environment of Network-2.2 are as detailed in Table 7. The average reward per time step is \( \approx 0.8 \), as observed in Figure 6. At test time (Figure 6 in manuscript), we observe a median error of \( \approx 500 \) pixels for centering and 0.2 m error in 3-D pose estimation.

### 9.3 Network 2.3: Centering, continuous collision avoidance and Multiview HMR Reward:

Table 8 showcases the environment parameters for training the final multi-agent network. Note that, in contrast to other multi-agent networks proposed, here, the actor is moving at a velocity sampled from a normal distribution, and the goal for the actor go-to-goal controller is sampled from a uniform distribution. The network is trained for 4 million time steps. We observe a reward of \( \approx 200 \) at 4 million steps in Figure 7. At test-time, we observe a median centering error of 600 pixels.
and a pose estimation error of 0.3 m. We infer that the higher error in comparison to networks 2.1 and 2.2 is attributed to the motion of the actor and the resulting motion of the MAVs.

References

[1] R. Tallamraju, E. Price, R. Ludwig, K. Karlapalem, H. H. Bülthoff, M. Black, and A. Ahmad, “Active perception based formation control for multiple aerial vehicles,” *IEEE Robotics and Automation Letters*, 2019.

[2] E. Price, G. Lawless, R. Ludwig, I. Martinovic, H. H. Bülthoff, M. J. Black, and A. Ahmad, “Deep neural network-based cooperative visual tracking through multiple micro aerial vehicles,” *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3193–3200, Oct 2018.

[3] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, “High-dimensional continuous control using generalized advantage estimation,” *arXiv preprint arXiv:1506.02438*, 2015.

[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.

[5] J. Ho and S. Ermon, “Generative adversarial imitation learning,” in *Advances in neural information processing systems*, 2016, pp. 4565–4573.