2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

SelfieDroneStick: A Natural Interface for Quadcopter Photography

Saif Alabachi[1] Gita Sukthankar[2] Rahul Sukthankar[3]

***Abstract*— A physical selfie stick extends the user's reach, enabling the acquisition of personal photos that include more of the background scene. Similarly, a quadcopter can capture photos from vantage points unattainable by the user; but teleoperating a quadcopter to good viewpoints is a difficult task. This paper presents a natural interface for quadcopter photography, the *SelfieDroneStick* that allows the user to guide the quadcopter to the optimal vantage point based on the phone's sensors. Users specify the composition of their desired long-range selfies using their smartphone, and the quadcopter autonomously flies to a sequence of vantage points from where the desired shots can be taken. The robot controller is trained from a combination of real-world images and simulated flight data. This paper describes two key innovations required to deploy deep reinforcement learning models on a real robot: 1) an abstract state representation for transferring learning from simulation to the hardware platform, and 2) reward shaping and staging paradigms for training the controller. Both of these improvements were found to be essential in learning a robot controller from simulation that transfers successfully to the real robot.**

I. Introduction

Although there has been work on improving quadcopter teleoperation [1], [2] and robot-assisted photography [3], [4], the premise behind most of these investigations has been that the user must learn the proposed interface paradigm. Our philosophy is to make the users learn as little as possible and the system learn as much as necessary. The *SelfieDroneStick* interface mimics the functionality of a selfie stick, enabling the user to control the quadcopter with only a mobile phone by a simple gesture and a click as shown in Fig. 1.

The goal is to generate a well-framed selfie of the user against the desired background, as if it were taken using a virtual selfie stick extending from the user in the direction of the handheld smart mobile device (SMD). The user specifies the desired composition by taking an ordinary selfie using the SMD, where the relative orientation directly specifies the azimuth of the vantage point while the height, 3D space position, and desired distance is indirectly specified by the SMD elevation, position and size of the user's face in the captured frame respectively. The drone flies to the target viewpoint based on the vantage point specified by the SMD to capture the selfie using a learned controller. The drone mirrors the bearing of the SMD as measured by its onboard IMU and selects an appropriate distance such that the user's body visually occupies relatively the same area in the drone selfie as the user's face did in the SMD frame. The resulting photos frame the user against the entire background, just as if the user had used a impossibly long selfie stick to compose the photograph.

Instead of attempting to use deep reinforcement learning (RL) to learn a direct control policy based on the raw pixel data as was done in [5], our controller utilizes an abstract state space representation. First, our perception system, Dense Upscaled Network (DUNet) [6], is trained to detect a human face (which is prominent in the phone camera image) and also the human body (visible from the drone's viewpoint). Deep Deterministic Policy Gradient (DDPG) [7] is then used to learn the flight policy in simulation using an abstract, continuous state space before being transferred to the real robot. To create a smooth and steady flight trajectory, we shape the reward to take into account both position and velocity.

This work introduces a novel interface for automating UAV selfie-photography using a mobile device. Our system takes selfies at the specified depth, background and orientation angle in the scene, with the user placed at the desired position in the frame. Once trained, our RL controller autonomously flies the quadcopter to the user-selected vantage point. Creating a selfie using our system typically takes 5s for the user to compose the shot and 8s for the drone to fly the maneuver, vs. 60s to manually operate the drone for a similar shot. Our system architecture is shown in Figure 1, and the ROS configuration, simulated environment, and code are publicly available.[1]

II. Related Work

Natural user interfaces (NUI) rely on innate human actions such as gesture and voice commands for all human-robot interaction [8], [9], [10]. Alternatively, more precise navigation in indoor and outdoor environments can be achieved through structured waypoint designation strategies [2], [11]. Wearable sensors were employed in a point-to-target interaction scenario to control and land a drone using arm position and absolute orientation based on the inertial measurement unit (IMU) readings [12]. Our system removes the need to employ gestures, hand crafted strokes, or wearable devices. Any mobile device equipped with a camera and IMU sensors can be used to direct the quadcopter using our *SelfieDroneStick* interface.

A subset of the human-robot interaction research has specifically addressed the problem of user interfaces for drone-mounted cameras. For instance, [2] tracks user-specified objects with an adaptive correlation filter in order to create photo collections that include a diversity of

[1]Saif Alabachi is with the University of Technology, Baghdad, Iraq s.ghassan@gmail.com

[2]Gita Sukthankar is with the Department of Computer Science, University of Central Florida, Orlando, FL gitars@eecs.ucf.edu

[3]Rahul Sukthankar is with Google sukthankar@google.com

[1]https://github.com/cyberphantom/Selfie-Drone-Stick

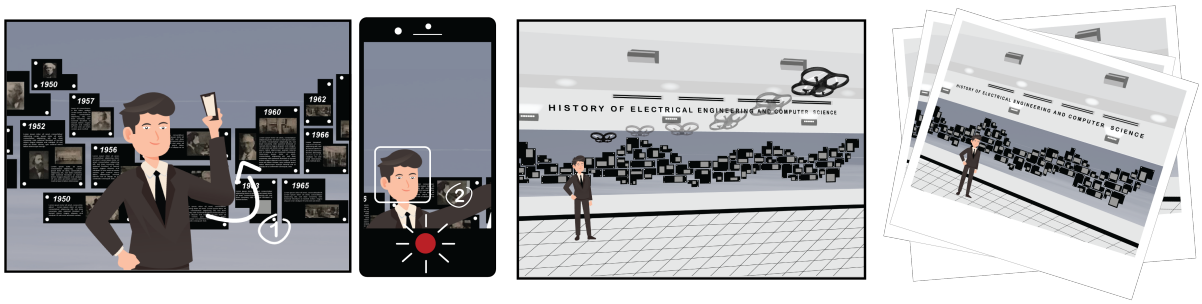

Fig. 1. *SelfieDroneStick* enables long-range selfie photography by allowing the user to naturally specify vantage points from which a drone can capture well-framed photos of the user.

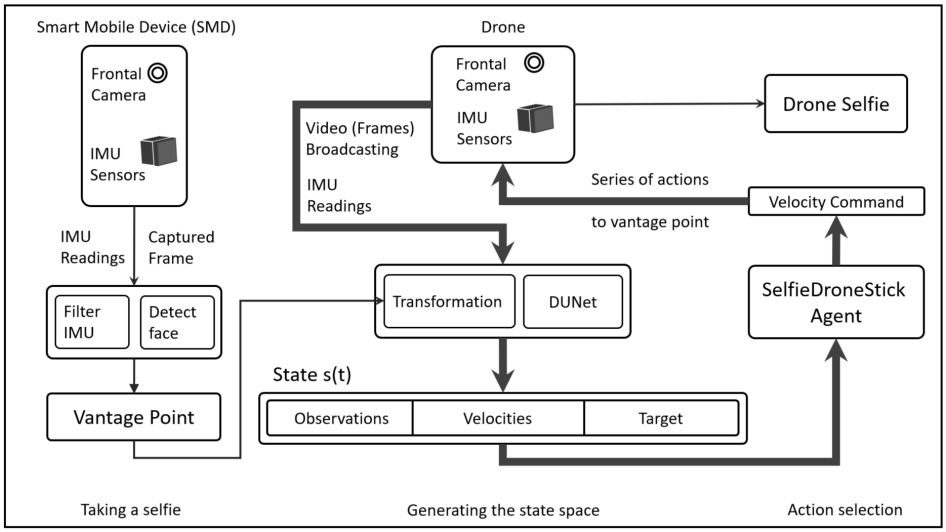

Fig. 2. Using the device camera, the user snaps the shot as if taking a selfie while angling the phone to modify the face size and background in the captured frame. The camera frame and IMU sensor readings are then extracted to generate the desired vantage point. DUNet is employed to detect the human face and body in both the images captured by the mobile device and the drone. The transformation module creates the target point, and the SelfieDroneStick software agent autonomously navigates there using the learned control policy.

viewpoints. XPose [1] is a touch-based system for photo taking in which the user concentrates on adjusting the desired photo rather than the quadcopter flight path. XPose offers an innovative, powerful user interface for taking a variety of photos, supported by trajectory planning. For our research, we chose to focus on the most popular photo composition (selfie in front of sweeping background), while using deep learning to make the vantage selection process fast and intuitive for the user.

Deep RL has been used to learn specialized flight controllers; for instance, Deep Q-Network (DQN) was used to learn autonomous landing policies for a quadcopter with a downward facing camera [13]. DQN also has been employed for capturing frontal facial photos; Passalis and Tefas [14] developed a realistic simulation for this task and trained their system on a database of head positions. In order to use DQN, both these systems adopted a discretized action space. In our system, we compare the performance of DQN, PG (policy gradient), DDPG (deep deterministic policy gradient) and found that the DDPG produced trajectories with less oscillation, particularly when combined with our reward shaping method. Rodriguez et al. [15] demonstrated good results with a similar approach on the challenging task on autonomous multirotor landing on a moving platform. In order to capture the perfect selfie, our learned controller must be able to execute multiple types of flight paths and is not limited to a single maneuver.

III. Method

Fig. 2 presents an overview of the *SelfieDroneStick* system. First, the user specifies vantage points for the drone using an SMD, simply by clicking a series of selfies. For each vantage point, the system captures both a reference camera image as well as the SMD's corresponding orientation from its inertial measurement unit (IMU) sensor. By combining the orientation of the SMD with the framing of the user's face in the SMD camera image, we can extrapolate an ideal vantage point for the drone. This is transformed into a desired framing for the user in the drone's camera. Next, on the drone we combine information from its onboard

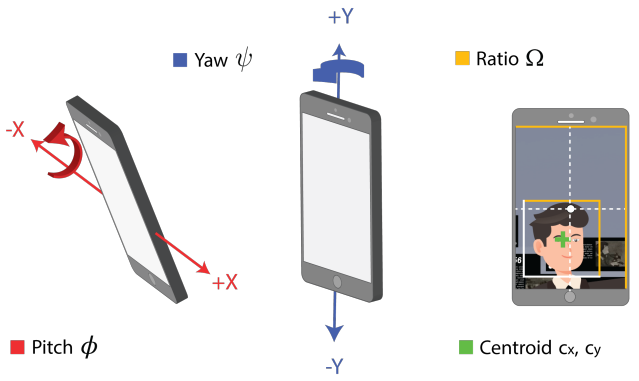

Fig. 3. The drone vantage point is specified by the orientation of the smart mobile device, ϕ and ψ, as well as the position and size of the user in the camera frame, (c_x, c_y) and ω. Moving the SMD away from the user corresponds to extending a "virtual selfie stick", guiding the drone to capture a photo from further away.

IMU, its front-facing camera and the vantage point. We employ a fast object detector to localize the user in each frame (operating at frame rate is essential, particularly in a crowded scene) and form a state vector that is used by the SelfieDroneStick reinforcement learning (RL) agent to plan the drone trajectory to the next vantage point. Finally, the RL agent, which was trained in simulation, controls the drone via a series of velocity commands to guide it through the sequence of vantage points. When the drone reaches each vantage point, it captures a long-range selfie of the user. The following sections discuss each of these steps in greater detail.

A. A Natural Interface for Vantage Point Specification

The user activates the SelfieDroneStick by taking a regular selfie using our web-based camera app. Then the shutter is pressed, the SMD's x-axis and y-axis orientation are recorded along with the camera image (Fig. 3). The IMU information partially specifies a bearing from the user along which the drone should seek to position itself in order to capture the desired shot. In addition to the bearing of the desired vantage point, we also need to specify the range. The key idea behind our interface is to enable the user to naturally specify the distance to the vantage point simply by varying the distance of the SMD from the user's face — moving the SMD further away should cause the drone to capture photos from further away. Intuitively, extending the user's arm corresponds to a (magnified) extension of a selfie stick. This requires us to localize where the user is located within the selfie frame. We employ the DUNet architecture [6], a real-time object detection CNN model, for face detection and the same model is also used for person detection on the drone camera (see below). The position of the user's face (c_x, c_y), along with the ratio of the face bounding box to camera frame ω and the SMD orientation ϕ and ψ fully specify the drone vantage point; this is illustrated in Fig. 3.

How do these measurements translate to a specific drone position? Visualizing the drone as if it were mounted on a virtual selfie stick extrapolated along the user's arm, past the SMD, shows that some of the coordinates map directly from the SMD to the drone: for instance, the azimuth to the drone is simply the yaw angle of the SMD, ψ. Others require an explicit transform: we empirically observe that the ratio of the user's face to the SMD image size falls in the range of $[0.1, 0.2]$ while desirable drone shots have a ratio of user's body to drone image size of $[0.03, 0.4]$. Thus, we linearly map the ranges of the former to the latter to obtain suitable range distances for the drone. Finally, the height of the drone is derived from a combination of SMD tilt ϕ and position (c_x, c_y) and size ratio ω of the user's face in the frame using straightforward geometry.

B. Perception System

There have been recent successes of Deep Reinforcement Learning that train end-to-end directly from pixel inputs in the context of automated game playing. However, for real-world robotics, it is still generally more practical to build reinforcement learning (RL) planners on top of stable perception systems. We use a specialized object detector DUNet [6] that is trained for detecting small objects and optimized for inference on compute-constrained platforms.

Since RL planners are trained in simulation, we employ an instance of DUNet pre-trained on data collected in the simulated world during training and replace it with a DUNet model pre-trained on PASCAL VOC [16] for real-world deployment. This enables the RL policy learned during simulation to transfer to the real-world without requiring explicit domain transfer. Since DUNet runs at real-time, we are able to perform per-frame detection while tracking the user through the stream of images. When the scene contains multiple people, it is important for the system to maintain its focus on the user; this is significantly easier in our application compared to the general tracking problem since the user is compliant and looking at the drone rather than averting their face or hiding behind obstacles. DUNet localizes detections with a bounding box and for the purposes of the RL planner, only the size and location of bounding boxes containing the user are relevant.

The drone's state space s_t composed of: observations – odometry derived from on-board sensors (gyroscope, accelerometer and pressure readings) and localization of the user in the drone's front-facing camera (generated by DUNet), (b) linear and angular velocities, (c) target – vantage point specification.

When designing a state space, it is worthwhile to focus only on the relevant features since RL scales poorly with state space dimensionality. For this reason, we condense (a) to a 5-D tuple that specifies the drone's pose relative to the user: $\mathbf{a} = [\psi^d, \Upsilon^d, c_x^d, c_y^d, \omega^d]$, corresponding to the current azimuth to the user, drone height, and observed bounding box location and size ratio, respectively; the superscript ^{d} denotes that these are all measurements on the drone rather than similar parameters measured on the SMD.

The velocity state (b) is straightforward, $\mathbf{b} = [\dot{x}, \dot{y}, \dot{z}, \dot{Z}$ consisting of three linear velocity components and the an-

gular velocity around the vertical axis (drone yaw rate of change).

The final aspect of the state (c) is the location of the next vantage point, specified using the same coordinates as the drone pose: $\mathbf{c} = [\psi^v, \Upsilon^v, c_x^v, c_y^v, \omega^v]$, as above with superscript ^{v} denoting that this specifies the vantage point.

The complete state vector is a concatenation of these three tuples, $\mathbf{s}_t = [\mathbf{a}, \mathbf{b}, \mathbf{c}]$, resulting in a 14-dimensional state space.

C. Learning a Deep RL Robot Control Policy: Rewards

At a high level, the goal of the *SelfieStickDrone* agent is to pilot the drone quickly and smoothly to each vantage point, without overshooting or oscillating. A naive formulation of such a goal in reinforcement learning (RL) would be to place sparse rewards only at the vantage points. Such an approach can work for simple problems in simulated environments but is challenging on our task because RL systems are inherently high in sample complexity and this is exacerbated by environments with sparse rewards (confirmed in our experiments below).

Practical RL for robotics relies heavily on training RL policies in simulation and then transferring the learned models to the real world. It also benefits significantly from *reward shaping* and curriculum learning through *staging* of rewards. We describe these in the context of our application.

Reward shaping for RL requires a careful balance between terms that are so punishing that they drive agents to absorption states (e.g., penalizing the agent at each time step in order to incentivize efficient flights could encourage the agent to end its episode quickly by crashing) and reward functions that are too rewarding near the goal that the agent chooses to dawdle near the goal state, accumulating a long sequence of partial rewards without achieving its objective.

Our reward function consists of two main terms: (1) a basin of attraction surrounding the specified vantage point to incentivize policies that fly the drone to the goal, and (2) a term to punish high-speed flight near the vantage point to encourage a smooth, non-oscillating approach before taking the selfie. The reward function also considers the fact that the drone flies through a sequence of vantage points and must smoothly transition from one to the next.

While creating a good reward function required considerable experimentation, we can explain its construction intuitively as follows. A natural expression for distance from the current drone pose to the next vantage point (goal) is given by the Euclidean distance between the corresponding pose vectors: $||\mathbf{a} - \mathbf{c}||_2 = ||(\psi^v, \Upsilon^v, c_x^v, c_y^v, \omega^v) - (\psi^d, \Upsilon^d, c_x^d, c_y^d, \omega^d)||_2$. We want the reward to decay with distance from the goal, and we also want to penalize speed when near the goal. From these, we propose the following formulation for the reward:

$$R = \text{CLIP}_{[0,1]}\left(\cos(\gamma||\mathbf{a}-\mathbf{c}||_2 e^{-\alpha||\mathbf{a}-\mathbf{c}||_2}) e^{-\beta||\mathbf{b}||_1}\right), \quad (1)$$

where $\text{CLIP}_{[0,1]}(.) = \max(\min(., 1), 0)$, $\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ are the three components of the RL state vector (described above) and $\{\alpha = 1.3, \beta = 0.35, \gamma = 11.3\}$ are empirically tuned hyperparameters. The $\cos(.)$ term bounds the basin around the vantage point where the agent can collect partial rewards and high speeds are not rewarded near the goal.

Each episode runs until one of the following termination conditions is triggered:

- If the agent achieves a reward of > 0.85 in a given timestep, the episode is terminated as an early success, with a reward $+1$.
- If the drone flies outside the safe zone (exceeds height or ratio limits), the episode is terminated as an early failure, with reward -0.8.
- If the object detector fails to find the subject (e.g., user is out of view), the episode is terminated early, with reward -0.8.
- If the step counter reaches the max. episode length (41 in our experiments), the episode terminates with the current reward, R.

At each time-step, the agent receives a reward R except for two cases, where the reward is explicitly shaped:

1) An exploration reward of $+1$ is given whenever the agent achieves $R > 0.75$; this is to encourage it to explore nearby states to achieve early success.
2) When the drone moves so that the detected person falls very close to the edge of the image (within 10 pixels of frame), the reward is set to -0.8, but the episode is not terminated early; unless the drone acts quickly, the person will go outside the frame and trigger the early condition discussed above.

The reward shaping incentivizes the agent to keep the user in its field of view and to stay within the safe zone. Once it can achieve these basic objectives, the agent can learn to fly the drone towards the vantage point.

A few other subtleties are worth mentioning: we provide the maximum reward $+1$ whenever the drone is within a radius of $1m$ of the vantage point and within $\pm 10^\circ$ in orientation. This acknowledges that the user's specification of the vantage point is intrinsically approximate and encourages the drone to fly quickly to the vantage point and take selfies while hovering rather than making unnecessary minor adjustments in an attempt to hit the exact position, which would have no meaningful impact on the qualities of photos acquired. The zero reward that the agent receives when it is far away from vantage points encourages the drone to fly quickly through those regions and the penalty for failing to detect the user is an incentive to keep the user in view (when possible). The latter is a form of curriculum learning: the drone first learns to explore while keeping the user in view and then learns to head to the vantage point – without losing the user.

D. Deep RL: Design Choices and Real-World Transfer

It is challenging to train a deep reinforcement agent to fly a real-world drone from scratch on this task. Therefore, we trained our system in a 3D emulator with a custom-built environment and a physical simulation of our quadcopter. The emulator creates a series of training simulations for

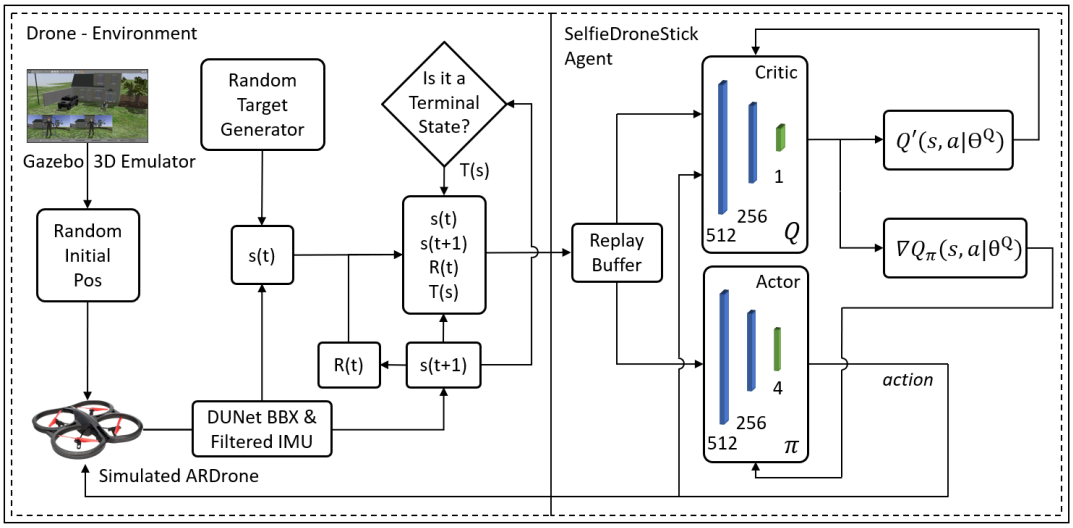

Fig. 4. Overview of the Deep RL agent training pipeline (target networks not shown). During each episode, the simulated drone is initialized in a random pose and assigned a random vantage point as target. The target critic network predicts the Q-value (Q') and the critic network provides the action gradients (∇Q_π).

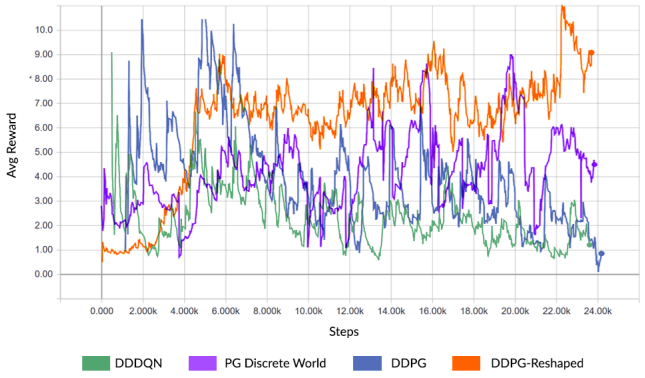

Fig. 5. Training progress for different Deep RL methods. DDPG with reward shaping improves steadily.

the SelfieDroneStick agent with different initialization poses and vantage points. The agent tries actions, observes a new state and collects rewards at each episodic time-step t. We explored training the agent with a variety of Deep RL algorithms (described below) but at a high level, the goal is to learn a policy $\pi(\mathbf{s})$ by updating a value function $Q_\pi(\mathbf{s}, \mathbf{a})$, where $\mathbf{s}$ and $\mathbf{a}$ denote the agent's current state and its selected action, respectively. Fig. 5 illustrates the learning progress for each of the tested methods.

Our action space is 4D and continuous: 3 linear velocities $(\dot{x}, \dot{y}, \dot{z})$ and a yaw rate $\dot{Z}$. Typical RL agents in the literature select an action from a discrete set (e.g., video game playing RL agents that select which button to press on a game controller) and learning in a multi-dimensional continuous action space, as is common in robotics, is more challenging.

Fig. 4 illustrates the training phase. We consider and evaluate several recent approaches for model-free Deep RL that have achieved state-of-the-art performance across relevant domains, as detailed below.

1) Dueling Double Deep Q Network (DDDQN): DDDQN is an updated variant on the popular Deep Q Network (DQN) [17]. Like DQN, this algorithm requires a discretization of the action space, for which we employ a simple 3x3x3 grid. The 'double' refers to the use of two independent networks to address the over-estimation of Q values in original DQN [18] and the 'dueling' splits the network into two separate parts [19]: one for estimating the advantage of selecting a given action among the others for the given state, and a second for estimating the state value.

2) Policy Gradient: Policy Gradient (PG) is an on policy approach for learning stochastic policies. In order to find the optimal policy, it increases the probabilities of actions that lead to higher return and reduces the probabilities of actions that lead to a lower return. During our evaluation of stochastic Gaussian policy gradient, we experienced convergence problems, accompanied by noisy gradients and high variance. Thus, we also considered the popular augmentation, Deep Deterministic Policy Gradient (DDPG) [7], which is based on the prior work on DPG [20].

3) Deep Deterministic Policy Gradient (DDPG): DDPG is an off-policy, actor-critic algorithm that yields good environment exploration through the use of a stochastic behavioral policy. The intuition is that it is easier to learn the optimal Q value by employing greedy deterministic policy learning through following the time difference (TD) bootstrapping error. The actor takes the state and predicts the action using its policy network, and the critic provides the Q value (expected return) based on the state and the actor's predicted action. Optimizing the Q value of the critic network is done by minimizing the loss between the prediction of the critic target network and the expected return. We implemented DDPG with a continuous action space that enables the drone to fly at a velocity of up to $4m/s$ in each direction.

4) DDPG with shaped reward: As above, but with reward shaping to encourage faster training. In the earlier variants, the reward function is ablated to consider only the sparse terms for vantage point proximity.

IV. Implementation Details and Experiments

Transferring the learned controller from simulation to the real world is simplified by our perception system and design of the state space. The Gazebo simulator and ROS were configured to work with the OpenAI Gym toolkit to observe a new state every 160ms. Training was performed on a single NVidia Titan X GPU. To expedite the training, we set threshold values for the human object ratio and drone height to generate a safe zone for the drone movement. The vertical space is set to be in the range of $[0.5, 3]$ meters and the yaw angle to be between $[-75°, 75°]$. To prevent the quadcopter from being too close to the detected human or moving too far and losing references, the human size ratio is only valid in the range $[0.5, 0.03]$.

Both the actor and critic networks have two hidden layers with sizes $[512, 256]$. The Adam optimizer is used for training with the learning rates set to 10^{-5} and 10^{-3} respectively. The output action selected by the actor is scaled to be in the range of $[-0.8, 0.8]$ as we are using a commodity quadcopter that exhibits shaky motion when flying faster than 0.8 ($\simeq 4m/s$) in any direction; this camera shake results in significantly degraded image quality and detection performance. The Deep RL agent is trained for 18K epochs in our simulated environment and then deployed on the drone.

During training, our environment consists of a 3D domain containing a single human and single UAV (drone) model against a simple background. We test the system in several realistic simulated scenarios, such as the one shown in upper part of Fig. 7. This allows us to conduct end-to-end experiments under repeatable conditions with known ground-truth, with the same perception system as we employ for real-world experiments, as shown in the lower part of Fig. 7.

Both the simulated and real-world scenarios follow a consistent script:

1) The drone is initialized facing the human subject at a safe distance. In simulation, take-off and landing are straightforward; for the real drone, the user initiates take-off by holding the SMD flat and initiates a landing sequence by tilting the SMD 90° around the x-axis.
2) Prior to activating the SelfieDroneStick, the drone hovers in front of the user, centering the user in the middle of the image with $\Omega_{obs} \simeq 24\%$.
3) Once the SelfieDroneStick agent has been activated by the user taking a selfie using the SMD, the SelfieDroneStick agent flies the drone to the specified vantage point using the learned Deep RL controller.
4) Once the drone arrives at the bearing and range consistent with the specified vantage point, it takes a long-range selfie of the subject.

Fig. 6 illustrates drone trajectories generated by the proposed controller (DDPG-reshaped) against two baselines, a classical PID contoller and a DDPG controller trained on sparse rewards. We see top and side views (rows) for each of four scenarios (columns) as the drone approaches the vantage point (green sphere). The shaped reward is better at slowing the drone to a stable hover at the vantage point. DDPG with sparse rewards fails on the fourth scenario.

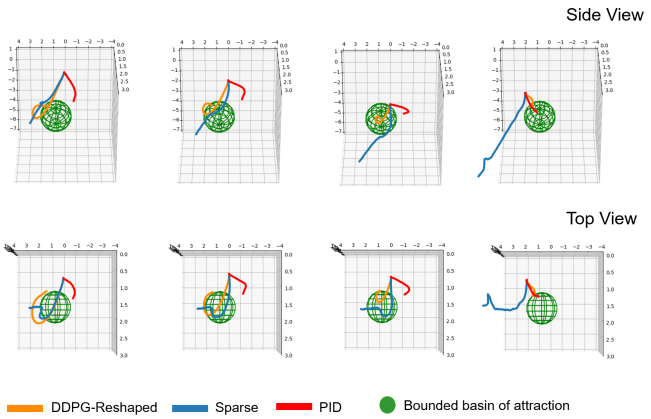

Fig. 6. Proposed controller (DDPG-reshaped) compared against a PID and DDPG trained on sparse rewards on 4 drone photography scenarios. The green sphere denotes target vantage point. See text for details.

TABLE I
Evaluation of proposed reward shaping vs. baseline

Scenario	Baseline Reward		Shaped Reward	
	Dist	Var. Vel	Dist	Var. Vel
1	0.133	0.168	0.144	**0.141**
2	0.235	0.0707	**0.148**	0.148
3	0.089	0.155	0.100	**0.147**
4	0.124	0.170	**0.085**	**0.144**

To better understand the effects of reward shaping, we compared the proposed shaped reward against a baseline reward that varies linearly with distance to vantage point. We measure both the distance to target and the variance in velocity over 10 timesteps to assess the controller's ability to bring the drone to the vantage point and its stability, respectively (see Table I).

Finally, Fig. 7 shows the *SelfieDroneStick* interface operating in three real-world scenarios. These experiments employed an iPhone SMD in conjunction with an AR.DRONE 2.0 UAV with a 30fps frame rate, with video streamed to a laptop that runs percepion and control. The interface enables the user to take multiple selfies with different backgrounds as the user moves in the environment. Videos of the system can be viewed at http://ial.eecs.ucf.edu/SelfieDroneStick/.

V. Conclusion

This paper introduces the SelfieDroneStick, our autonomous navigation and selfie-photography platform that takes long-range selfies using a drone from vantage points specified by the user using a natural "virtual selfie stick" interface. Designing the SelfieDroneStick interface required overcoming several significant challenges: (1) specifying the composition of desired selfies using the smartphone, (2)

Fig. 7. Evaluating *SelfieDroneStick* in a custom simulation environment and in real-world using the Parrot ARDrone 2.0.

learning Deep RL policies that transfer from simulation to the real world robustly, (3) ensuring that perception, cognition and control operate on compute-constrained platforms at frame-rate. Our experiments in simulation and on the quadcopter confirm the feasibility of creating a natural interface for quadcopter photography driven by a learned RL policy.

ACKNOWLEDGMENTS

We thank Yasmeen Alhamdan for help in figure generation.

REFERENCES

[1] Z. Lan, M. Shridhar, D. Hsu, and S. Zhao, "Xpose: Reinventing user interaction with flying cameras," in *Robotics: Science and Systems*, 2017.

[2] S. Alabachi and G. Sukthankar, "Intelligently assisting human-guided quadcopter photography," in *Proceedings of FLAIRS*, 2018.

[3] E. Cheng, *Aerial photography and videography using drones*. Peachpit Press, 2015.

[4] R. Coaguila, G. Sukthankar, and R. Sukthankar, "Selecting vantage points for an autonomous quadcopter videographer," in *Proceedings of FLAIRS*, 2016, pp. 386–391.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[6] S. Alabachi, G. Sukthankar, and R. Sukthankar, "Customizing object detectors for indoor robots," in *International Conference on Robotics and Automation*, 2019.

[7] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[8] V. L. Popov, K. B. Shiev, A. V. Topalov, N. G. Shakev, and S. A. Ahmed, "Control of the flight of a small quadrotor using gestural interface," in *IEEE International Conference on Intelligent Systems*, 2016, pp. 622–628.

[9] R. A. Suárez Fernández, J. L. Sanchez-Lopez, C. Sampedro, H. Bavle, M. Molina, and P. Campoy, "Natural user interfaces for human-drone multi-modal interaction," in *International Conference on Unmanned Aircraft Systems*, 2016, pp. 1013–1022.

[10] L. Ma and L. L. Cheng, "Studies of AR Drone on gesture control," in *International Conference on Materials Engineering, Manufacturing Technology and Control*, 2016.

[11] C. Gebhardt, B. Hepp, T. Nägeli, S. Stevšić, and O. Hilliges, "Airways: Optimization-based planning of quadrotor trajectories according to high-level user goals," in *Proceedings of the CHI Conference on Human Factors in Computing Systems*, 2016, pp. 2508–2519.

[12] B. Gromov, L. Gambardella, and A. Giusti, "Video: Landing a drone with pointing gestures," in *ACM/IEEE International Conference on Human-Robot Interaction Companion*, 2018.

[13] R. Polvara, M. Patacchiola, S. Sharma, J. Wan, A. Manning, R. Sutton, and A. Cangelosi, "Autonomous quadrotor landing using deep reinforcement learning," *arXiv preprint arXiv:1709.03339*, 2017.

[14] N. Passalis and A. Tefas, "Deep reinforcement learning for frontal view person shooting using drones," in *2018 IEEE Conference on Evolving and Adaptive Intelligent Systems (EAIS)*, 2018, pp. 1–8.

[15] A. Rodriguez-Ramos, C. Sampedro, H. Bavle, I. G. Moreno, and P. Campoy, "A deep reinforcement learning technique for vision-based autonomous multirotor landing on a moving platform," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1010–1017.

[16] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL visual object classes challenge: A retrospective," *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, Jan. 2015.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[18] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *AAAI*, 2016.

[19] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[20] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International Conference on Machine Learning*, 2014.