APPENDIX

A. Jackal Robot Experiments

The Jackal robot is equipped with a 5MP camera using a wide angle lens, an IMU sensor and an indoor Hokuyo UTM-30LX LIDAR sensor with a 270° scanning range and 0.1 to 10 meters scanning distance. The objective is to drive the robot using a camera in the center of a lane marked with tape using only data collected in the real-world. Training directly from real world experience requires addressing multiple issues such as minimizing wear and tear on the robot, and the need of human supervision during training in order to prevent or resolve robot crashes and recharge the battery.

There are two learned controllers, called GVF-BCQ and E2E-BCQ respectively, and one classical baseline called MPC (model predictive control). The learned controllers output a steering angle a_t^{steer} and target speed a_t^{speed} based on the image taken by the camera in order drive centered in a closed loop road outlined with tape on a carpeted floor. The MPC outputs a steering angle a_t^{steer} and target speed a_t^{speed} based on localization of the robot on a prior constructed map of the environment used to follow a sequence of waypoints supplied to the robot beforehand. Localization, map and waypoints are needed to train the GVF-BCQ controller; however, this information is no longer used during testing. GVF-BCQ tolerates noisy, low-accuracy localization methods that might otherwise be not accurate enough for smooth control with MPC or other methods.

1) Training and testing environment: The environment used for collecting data and evaluating the agents included two types of carpet floor - each with a different amount of friction. The evaluation roads were done on one carpet only which was included in only about 20% of the training data; the rest of the training data was on another type of carpet flooring to provide a generalization challenge for the learned controllers. The friction was quite high on both carpets and it caused the agent to shake the camera violently while turning since the robot employs differential steering; tape on the wheels helped reduce the friction a bit. Nevertheless, localization techniques using wheel odometry was deemed unsuitable and LIDAR-based localization was used instead. LIDAR localization was not highly accurate but was sufficient; our tests showed that it was repeatable to within roughly 5 centimeters which is deviation of upto about 13% from the center of the road.

Nine closed loop roads were created by placing left and right lanes markings on the carpeted floor separated to form a consistent lane width of roughly 76 centimeters for all the roads; some error in lane width measurements were tolerated when creating our roads. 6 of the roads were selected for collecting data to train our learned agents. Data was collected in both directions. The remaining 3 roads were reserved for testing the performance of the policies. In addition, each test road included damaged variants for a total of 6 test roads.

The poses and orientations of a sequence of waypoints were established to denote the center of lane which is the desired path of the agent; this was used to train our agents on the training roads and evaluate them on the test roads. The center waypoints were collected by an expert manually and carefully navigating the Jackal robot on the road in the approximate center of lane; while this was inaccurate, it was not found to harm learning since the GVF-BCQ approach was able to generalize and learn features to drive the vehicle centered in the lane. The LIDAR-based localization produced poses periodically to form the center waypoints; these were cleaned up by removing overlapping waypoints to form a closed loop path. However, it did not produce poses at a consistent sampling frequency and thus a linear interpolation method was used to fill in the gaps and provide localization and center waypoint information at every time step. The purpose of the center waypoints was to compute the road angle and lane centeredness of the robot in the road at any given time which is needed to train the GVF predictions and evaluate all of our controllers.

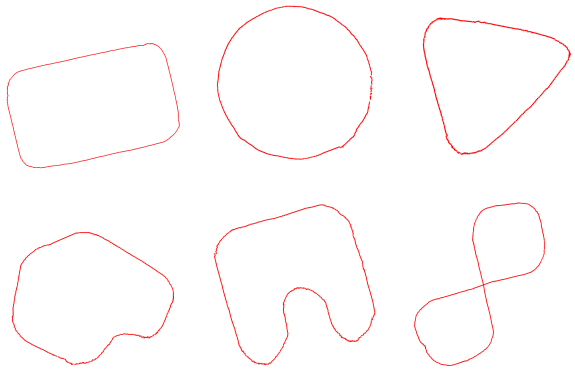


Fig. 7: Six roads for training. The second row shows more complex road structure. The rectangle road has rectangular edges at all corners.

The center waypoints for the training and testing roads are depicted in Figure 7 and Figure 8, respectively. The first three training roads were simple geometric shapes while the other three were a bit more complex. The first test road was the most similar to the training data where the outer edge of the four corners were rounded. The second test road was an oval shape to evaluate how well the agent maintained a turn that, unlike the circle training road, requires the steering angle to be modulated rather than remain constant. The third test road was a complex shape with multiple sudden turns that was very different from any of the roads in the training data set. This tests generalization to new roads and confirms that the agent is not memorizing an action sequence to remain centered in the path. All methods were evaluated at 0.25 m/s and 0.4 m/s maximum speeds and clock-wise (CW) and counter clock-wise (CCW) directions. In order to test robustness, all three test roads were altered by degrading or damaging the lane markings. The complex test road also included distracting markings as shown in Figure 1. An example of an image received from the robot with and without damage to the lane markers is shown in Figure 9.

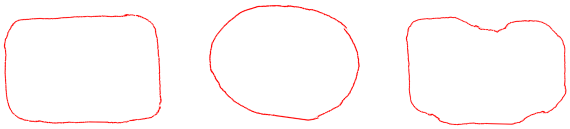


Fig. 8: Three roads for testing. Left to right: (1) rectangle with rounded corners; (2) oval; (3) complex.

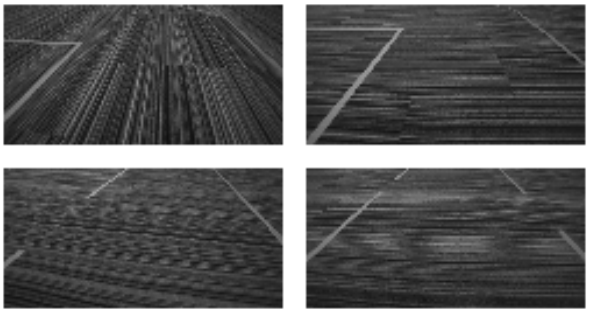


Fig. 9: Input images of normal lane markings (top row) and damaged lane markers (bottom row).

2) Data collection: Both learned agents – GVF-BCQ and E2E-BCQ – were trained with batch RL [23] where the GVF-BCQ method learned a predictive representation. Rosbags were collected with the Jackal robot each containing around 30 minutes to 2 hours of data for a total of 40 hours of data containing approximately 1.5 million images. The camera image, localization pose, IMU measurement, and action taken by the controller was recorded at each time step. The Jackal robot was controlled using a random walk controller that was confined to the road area to provide sufficient and safe exploration. The map was created with Hector SLAM [55] and the localization produced by Adaptive Monte-Carlo Localization [56].

The random walk controller was based on a pure pursuit controller, where action taken at each time step is defined by

$$\begin{aligned} a_t^{steer} &= \text{clip}(\text{angle}(p_t, p^*_{k(t)}) - \theta^z_t, -\pi/2, \pi/2) \\ a_t^{speed} &= \text{clip}(v^*_{k(t)}, 0.2, 0.5) \end{aligned} \tag{9}$$

where θ^z_t and p_t were the yaw angle and the 2-dimensional position of the robot in the real world (obtained from localization), $p^*_{k(t)}$ and $v^*_{k(t)}$ were the target position and linear velocity at time t, clip(x, MinVal, MaxVal) is the clip function that operates in scalar and vector element-wise, and angle$(p_t, p^*_{k(t)})$ is the function that returns the yaw angle in the real world of a vector that points from p_t to $p^*_{k(t)}$. The target position $p^*_{k(t)}$ and linear velocity $v^*_{k(t)}$ were encapsulated in the next target pose at index $k(t)$ in the sequence of target poses:

$$\begin{aligned} k(1) &= 1 \\ k(t+1) &= \begin{cases} k(t)+1 \text{ if } ||p_t - p^*_{k(t)}||_2 < 0.025 \\ k(t) \text{ otherwise} \end{cases} \end{aligned} \tag{10}$$

Thus, the robot advanced to the next target position and linear velocity in the target pose sequence once it arrived within 2.5 centimeters of the current target position. In order to provide efficient and safe exploration that can be confined to the road area, the target position p^*_j was based on the position $\tilde{p}_j$ of the center waypoints collected earlier with some noise added to provide the necessary exploration to learn the predictions and policy:

$$\begin{aligned} p^*_j &= \tilde{p}_{j\%N} + \varepsilon^p_j \\ v^*_j &= \begin{cases} v^*_{j-1} + \varepsilon^v_j \text{ if } j > 1 \\ 0.35 \text{ if } j = 1 \end{cases} \end{aligned} \tag{11}$$

where N is the number of points that define the center waypoints of the closed loop road. ε^p_j and ε^v_j were the noises added at each time step:

$$\begin{aligned} \varepsilon^p_j &= \begin{cases} \text{clip}(\varepsilon^p_{j-1} + \mathcal{N}(0, 0.02 * \mathbb{1}), -0.3, 0.3) \text{ if } j > 1 \\ [0,0]^\intercal \text{ if } j = 1 \end{cases} \\ \varepsilon^v_j &= \mathcal{N}(0, 0.02) \end{aligned} \tag{12}$$

The noises for the poses were clipped so that the robot would not explore too far outside the road area.

The rosbags were processed to synchronize the sensor data streams at a fixed sample frequency of 10Hz and compute the lane centeredness α_t, road angle β_t, and speed v_t of the robot at each time step:

$$\begin{aligned} \nu_t &= \text{knn}(p_t, S_t) \\ \alpha_t &= \text{clip}(\frac{||p_t - \nu_t||_2}{H}, -1.0, 1.0) \\ \beta_t &= \text{clip}(\text{angle}(p_t, \nu_t) - \theta^z_t, -\pi/2, \pi/2) \end{aligned} \tag{13}$$

where knn(x, S) returns ν as the closest point to x in S using k-nearest neighbor and $H = 38$ centimeters as the half lane width. S_t is a pruned set of center waypoints where $S_t = \{\tilde{p}_t\}$ for all roads, except for the figure 8 road in the lower right of Figure 7 where S_t was based on a sliding window to prevent issues with k-nn at the intersection:

$$S_t = \begin{cases} \{\tilde{p}_t\} \text{ if } j = 1 \\ \{\tilde{p}_{t=I_{t-1}-w \to I_{t-1}+w}\} \text{ if } j > 1 \end{cases} \tag{14}$$

where I_{j-1} is the index of ν_{t-1} in S_{t-1} at the previous time step and $w = 10$ is the size of the sliding window. Negative indices are wrapped to the beginning of the waypoint list. The speed was estimated using the change in position over a single time step which was quite noisy but more reliable than speed returned from the robot's odometry. Due to computation constraints on the robot, the localization messages were output at less than 10Hz; thus, a linear interpolation was used to fill in missing poses and orientations in the data and synchronize the data streams.

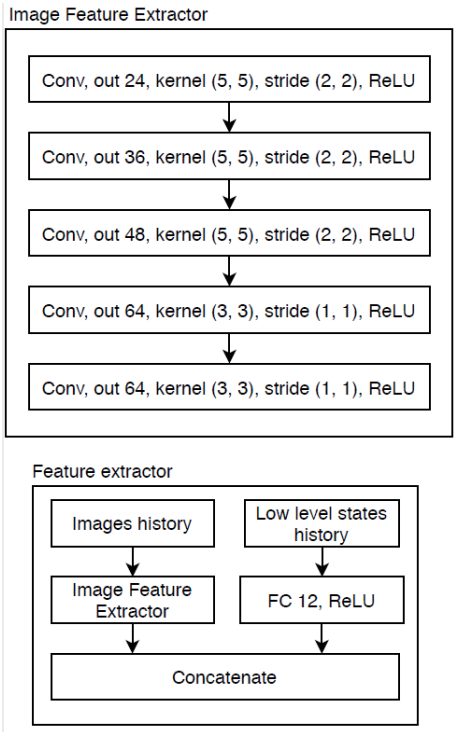
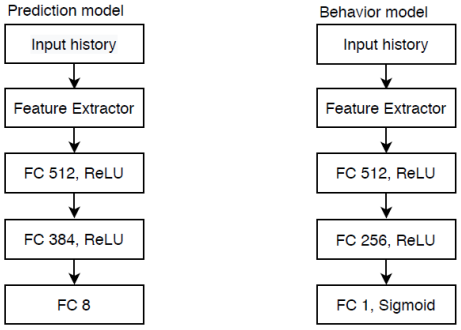


Fig. 10: Model of the feature extractor of the robot's state concatenates features from the image with low-dimensional state information from the robot like speed and last action.

Images from camera with original size 1920×1080 were cropped to 960×540 region in the center and then additionally top-cropped by 60. The images were then downsampled by a factor of 8 in both spatial dimensions to give a final size of 120×60 and then converted to gray scale. To improve generalization in deep learning and balance the left-right biases in the data, augmented data was created with horizontally flipped images along with the corresponding signs of the lane centeredness α_t, road angle β_t, and steering action a_t^{steer}) flipped.

3) GVF-BCQ Training: The predictive neural network used was trained using the offline version of the predictive learning algorithm 2. The transitions in the data were loaded into a replay buffer in the same order that the transitions were observed in the rosbag where mini-batches of size 128 where sampled from the growing replay buffer and used to update the GVFs. The GVFs were updated for 5 million steps followed by BCQ for an additional 5 million steps for a total of 10 million steps. The order that the rosbags were loaded into the replay buffer was randomized. The replay buffer had a maximum capacity of 0.5 million samples; once the replay buffer was filled, the oldest samples were removed. Training began once the replay buffer reached 0.1 million samples. While an alternative approach would have been to sample mini-batches from the entire data set from the beginning, our approach was found to be effective and required minimal changes to the data loader of the online version of the algorithm.



(a) Prediction Model (b) Behavior Model

Fig. 11: Neural network models for (a) Prediction Model that produces $\psi(s)$, and (b) Behavior Model that estimates $\mu(a|s)$

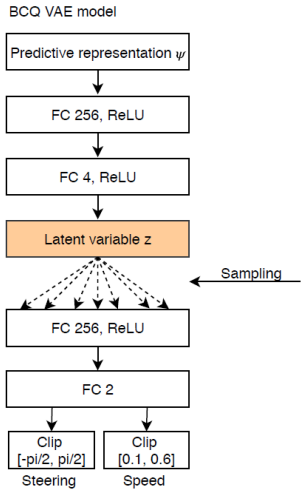
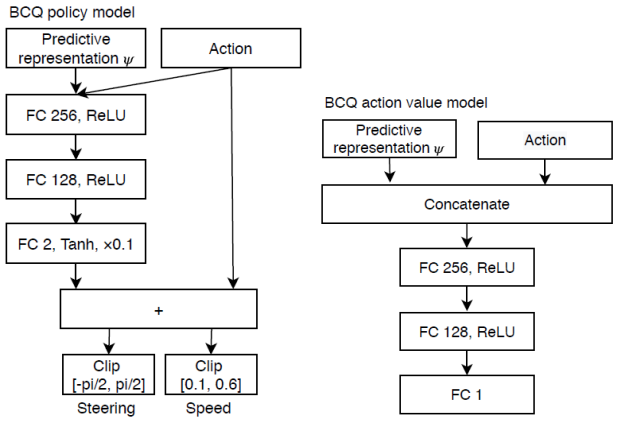


Fig. 12: Model of the GVF-BVQ variational auto-encoder (VAE).

The feature extractor model of the robot state is depicted in Figure 10. Estimating the behavior distribution for the GVF predictions was done with $\eta(a|s)$ as a uniform distribution defined on the interval $[-\pi/2, \pi/2]$ for the steering action and a uniform distribution defined on the interval $[0, 1]$ for the target speed action. The neural network model used to learn the GVFs predictions that produce ϕ are depicted in Figure 11a. The model used to estimate the behavior distribution $\mu(a|s)$ is shown in Figure 11b. The BCQ network models that are used to learn the policy of the agent were all relatively small fully connected networks with hidden layer of size 256 as shown in Figures 12, 13a, and 13b.

The predictive representation ψ is a vector of length 11 consisting of ϕ (vector of predictions of size 8), the last steering action, the last target speed action and the current robot speed. The latent vector dimension was 4 which was a Normal distribution parameterised by mean and log standard deviation. All networks used ReLU activation for the hidden layers and linear activation for the outputs. The action output from the actor and VAE were clipped to $[-\pi/2, \pi/2]$ for steering and $[0.1, 0.6]$ for the target speed.



(a) Actor Model (b) Critic Model

Fig. 13: Neural network models for GVF-BCQ (a) Actor, and (b) Critic

The weight of the KL divergence loss used in BCQ was 0.5. The learning rate was 10^{-4} for both GVF and BCQ model training. Figure 14 shows the training curves for the predictive representation (GVFs) including the temporal-difference loss, behavior model loss and mean importance sampling ratio.

4) End-to-end BCQ Baseline Training: Nearly the same training setup used for GVF-BCQ was also applied to the E2E-BCQ method. The hyperparameters, training settings, activation functions for the output and action clipping are exactly the same as GVF-BCQ unless noted otherwise. All the networks in E2E-BCQ including the VAE, actor and critic shared the same feature extractor as the GVF-BCQ shown in Figure 10. The neural network models are given in Figures 15b, and 15a.

For a fair comparison, E2E-BCQ was trained for 10 millions update steps - the same number as GVF-BCQ method which divides the update budget evenly where 5 million updates are applied to the GVF predictions and 5 million updates are applied to BCQ networks. The agent was then tested on the rectangle test road and it was found that E2E-BCQ performed very poorly. The agent was too slow reaching an average speed of about 0.18 m/s whereas the GVF-BCQ method was able to reach double that speed. In addition, E2E-BCQ steered very poorly and was often not centered in the lane; unlike the GVF-BCQ method which was observed to be quite robust, the E2E-BCQ method sometimes drove out of the lane where an emergency stop was needed to prevent collision. For this reason, E2E-BCQ was only compared to GVF-BCQ on the rectangle test road; and we focused on comparisons against the MPC controller which was more robust than E2E-BCQ. A detailed evaluation of E2E-BCQ is shown in Figure 18 and Table I.

5) MPC Baseline: An MPC baseline using standard ros nodes available for the Jackal robot were used for controlling the Jackal robot. The baseline was tuned for 0.4 m/s; however, it was challenging to achieve good performance due to limited computation power for the look ahead, noisy localization with LIDAR scan matching and inaccurate modeling of the steering characteristics on carpet floors. The best performance was achieved for 0.25 m/s but significant oscillation was observed for 0.4 m/s that was very challenging to completely eliminate. The center waypoints provided as input to the MPC controller were processed so that the minimum distance between two consecutive waypoints was 2.5cm; the waypoints were then downsampled by a factor of 16 in order to increase their separation and increase the look ahead distance; different downsampling factors was tested but oscillation was never completely eliminated. The MPC had an optimization window of 5 steps into the future; this was limited by computation available on the Jackal robot's on-board computer. This look ahead ensured the MPC was far enough into the future for real-time control.

6) Test Results: This section provides a detailed comparison of GVF-BCQ and the baselines at different speeds and directions. For both GVF-BCQ and E2E-BCQ methods, the actor produced the steering and desired speed and thus the agent was able to modulate its own speed and slow down as necessary in advance of sharp turns. The speed command was clipped at the maximum target speed. The controllers started at the same position and heading angle and they were allowed to run for exactly 300 seconds. The agents were evaluated based on the following criteria:

- Reward per second: $\frac{1}{N}\sum_{t=1}^{N} r_t$
- Average speed: $\frac{1}{N}\sum_{t=1}^{N} v_t$
- Average absolute lane centeredness: $\frac{1}{N}\sum_{t=1}^{N} |\alpha_t|$
- Average absolute road angle: $\frac{1}{N}\sum_{t=1}^{N} |\beta_t|$
- Near out of lane[3]: $\frac{1}{N}\sum_{t=1}^{N} \mathbb{1}_{|\alpha_t|>0.75}$[4].
- First Order Jerk[5]: $\frac{1}{N-1}\sum_{t=1}^{N-1} |a_{t+1} - a_t|$
- Second Order jerk[6]: $\frac{1}{N-2}\sum_{t=1}^{N-2} |(a_{t+2} - a_{t+1}) - (a_{t+1} - a_t)|$

A comparison of GVF-BCQ and E2E-BCQ is given in Tables IV and V. Experiments are named according to the method used, the selected target speed and the direction of the road loop (i.e. counter-clock-wise versus clock-wise). For example, GVF-BCQ-0.4-CCW points to the test of the GVF-BCQ controller with 0.4 m/s target speed in the counter-clock-wise direction.

Finally, the GVF-BCQ method generalized well to damaged lane markings and distractions in the visual images as shown in the similar scores and similar distributions in Figure 17 and Table VI and VII. Experiments on roads with damaged lane markings are denoted with suffix -D.

Details evaluations against the MPC baseline are shown in Tables VIII and IX. In our evaluation at 0.25 m/s in the counterclockwise direction, the gap between the controllers narrowed but GVF-BCQ still out-performed MPC overall. A

[3]ratio of time steps where the agent's absolute lane centeredness is greater than 0.75

[4]Where $\mathbb{1}$ is the indicator function.

[5]First order jerk is the absolute change of action taken by the agent in one time step. Lower jerk scores are better. Both steering and speed actions considered separately.

[6]Second order jerk is the absolute change of the first order jerk in one time step. Lower jerk scores are better. Both steering and speed actions considered separately.

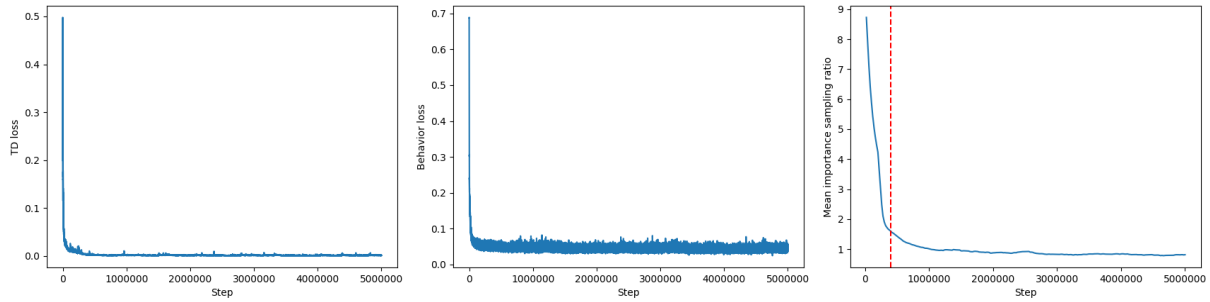


Fig. 14: TD loss, behavior loss and mean importance sampling ratio in the buffer over training steps. Red vertical dash line is the point when the buffer is full.

TABLE IV: Comparison of GVF-BCQ and E2E-BCQ on the Rectangle test road at 0.4 m/s

	Experiment	Reward per second ↑	Average speed ↑	Average off-center (normalized) ↓	Average off-angle ↓	Out of lane ↓
Rectangle shape	GVF-BCQ-0.4-CCW	**2.6835**	**0.3205**	**0.1345**	**0.1315**	**0.0%**
	E2E-BCQ-0.4-CCW	1.2578	0.1816	0.2558	0.2414	3.76%
	GVF-BCQ-0.4-CW	**2.2915**	**0.3140**	**0.2217**	**0.1586**	**0.0%**
	E2E-BCQ-0.4-CW	-0.1302	0.1710	0.9927	0.3034	54.18%

TABLE V: Comparison of GVF-BCQ and E2E-BCQ jerk levels on the Rectangle test road at 0.4 m/s

	Experiment	First order speed jerk ↓	Second order speed jerk ↓	First order steering jerk ↓	Second order steering jerk ↓
Rectangle shape	GVF-BCQ-0.4-CCW	**0.0356**	**0.2532**	**0.2251**	**1.3403**
	E2E-BCQ-0.4-CCW	0.0154	0.2266	0.1109	1.4240
	GVF-BCQ-0.4-CW	**0.0311**	**0.2149**	**0.1995**	**1.1850**
	E2E-BCQ-0.4-CW	0.0148	0.1937	0.1174	1.3514

TABLE VI: Evaluation of the robustness of GVF-BCQ method on damaged lane markings on all the test roads

	Experiment	Reward per second ↑	Average speed ↑	Average off-center (normalized) ↓	Average off-angle ↓	Out of lane ↓
Rectangle shape	GVF-BCQ-0.4-CCW	2.6835	0.3205	**0.1345**	**0.1315**	0.0%
	GVF-BCQ-0.4-CCW-D	**2.7407**	**0.3261**	0.1358	0.1351	0.0%
Oval shape	GVF-BCQ-0.4-CCW	**2.4046**	**0.3501**	**0.2754**	0.2125	**1.45%**
	GVF-BCQ-0.4-CCW-D	2.0728	0.3279	0.3285	**0.2089**	7.19%
Complex shape	GVF-BCQ-0.4-CCW	**2.3501**	0.3129	**0.2221**	**0.1817**	**0.0%**
	GVF-BCQ-0.4-CCW-D	2.1059	**0.3284**	0.3125	0.2365	9.42%

clear advantage of GVF-BCQ is the stability and smoothness of control achieved at the higher speeds. Our proposed GVF-BCQ method beats the MPC in reward and was better on all tracks at both speed values without access to localization information during testing. The reason for this can be explained by looking at the average lane centeredness of the agent. The MPC performed as well as the GVF-BCQ method while maintaining good speed; however, it fails at keeping the vehicle in the center of the lane. The reason may be due to a number of different factors including possible inaccuracies in the MPC forward model resulting from the friction between the wheels and the carpet in the test runs, especially at higher speeds. The MPC suffered from many near out of lane events and had trouble staying within the lane markings of the oval road. The GVF-BCQ method was better at controlling steering, leading to much higher average reward even though it had lower average speed at 0.4 m/s max speed. Additionally, the GVF-BCQ method was much better in achieving smooth control. These points are reflected in Figure 19 on the rectangle test track where the MPC lane centeredness distribution is skewed to one side and its steering action distribution has two modes that are far from zero while the GVF-BCQ method distributions are more concentrated around zero.

In order to provide more insight into the performance of the controllers, we also investigated the distributions of α_t, β_t, v_t and a_t in Figures 17, 18, and 19.

[6]Note that measured vehicle speed might not be equal to speed action from the agent due to physical constraints of the environment and noises in measurement.

TABLE VII: Evaluation of the jerk of GVF-BCQ method on damaged lane markings on all the test roads

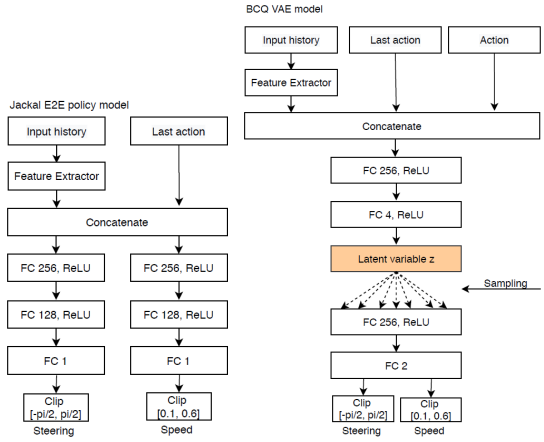
	Experiment	First order speed jerk ↓	Second order speed jerk ↓	First order steering jerk ↓	Second order steering jerk ↓
Rectangle shape	GVF-BCQ-0.4-CCW	**0.0356**	**0.2532**	**0.2251**	**1.3403**
	GVF-BCQ-0.4-CCW-D	0.0383	0.2715	0.2303	1.4620
Oval shape	GVF-BCQ-0.4-CCW	0.0348	**0.2423**	0.2191	**1.4632**
	GVF-BCQ-0.4-CCW-D	**0.0334**	0.2953	**0.2094**	1.6612
Complex shape	GVF-BCQ-0.4-CCW	**0.0341**	**0.2540**	**0.2272**	**1.5306**
	GVF-BCQ-0.4-CCW-D	0.0437	0.3608	0.2897	2.0946

TABLE VIII: Comparison of GVF-BCQ method and MPC on all the test roads at different speeds and directions

	Experiment	Reward per second ↑	Average speed ↑	Average off-center (normalized) ↓	Average off-angle ↓	Out of lane ↓
Rectangle shape	GVF-BCQ-0.4-CCW	**2.6835**	0.3205	**0.1345**	**0.1315**	**0.0%**
	MPC-0.4-CCW	0.9700	**0.3833**	0.5252	0.1943	20.42%
	GVF-BCQ-0.4-CW	**2.2915**	0.3140	**0.2217**	**0.1586**	**0.0%**
	MPC-0.4-CW	0.1282	**0.3836**	0.9086	0.1916	67.86%
	GVF-BCQ-0.25-CCW	**2.1442**	**0.2467**	**0.1098**	**0.1181**	0.0%
	MPC-0.25-CCW	1.1971	0.2412	0.1218	0.1308	0.0%
Oval shape	GVF-BCQ-0.4-CCW	**2.4046**	0.3501	**0.2754**	0.2125	**1.45%**
	MPC-0.4-CCW	0.8928	**0.3825**	0.5293	**0.1963**	22.75%
	GVF-BCQ-0.4-CW	**2.4848**	0.3658	**0.2953**	**0.1922**	**0.0%**
	MPC-0.4-CW	-0.7168	**0.3836**	1.3182	0.2095	91.22%
	GVF-BCQ-0.25-CCW	**1.5112**	**0.2473**	**0.3645**	0.1466	**3.32%**
	MPC-0.25-CCW	0.0225	0.2296	0.9565	**0.1381**	87.92%
Complex shape	GVF-BCQ-0.4-CCW	**2.3501**	0.3129	**0.2221**	**0.1817**	**0.0%**
	MPC-0.4-CCW	0.7172	**0.3845**	0.6407	0.2131	38.94%
	GVF-BCQ-0.4-CW	**2.3182**	0.3168	**0.2317**	**0.2150**	**0.06%**
	MPC-0.4-CW	0.4324	**0.3905**	0.7662	0.2264	52.23%
	GVF-BCQ-0.25-CCW	**1.9326**	**0.2472**	0.1890	**0.1509**	0.0%
	MPC-0.25-CCW	1.1559	0.2435	**0.1664**	0.1720	0.0%

TABLE IX: Comparison of jerk of GVF-BCQ method and MPC on all the test roads at different speeds and directions

	Experiment	First order speed jerk ↓	Second order speed jerk ↓	First order steering jerk ↓	Second order jerk jerk ↓
Rectangle shape	GVF-BCQ-0.4-CCW	**0.0356**	**0.2532**	**0.2251**	**1.3403**
	MPC-0.4-CCW	0.0832	0.7605	1.2542	8.1963
	GVF-BCQ-0.4-CW	**0.0311**	**0.2149**	**0.1995**	**1.1850**
	MPC-0.4-CW	0.0944	0.8916	1.4328	10.9570
	GVF-BCQ-0.25-CCW	**0.0009**	**0.0112**	**0.1466**	**0.8890**
	MPC-0.25-CCW	0.0570	0.5272	0.6384	3.5208
Oval shape	GVF-BCQ-0.4-CCW	**0.0348**	**0.2423**	**0.2191**	**1.4632**
	MPC-0.4-CCW	0.1026	0.9301	1.4119	8.9051
	GVF-BCQ-0.4-CW	**0.0241**	**0.1638**	**0.1674**	**1.1451**
	MPC-0.4-CW	0.0847	0.7534	1.3957	9.0432
	GVF-BCQ-0.25-CCW	**0.0005**	**0.0061**	**0.0969**	**0.7614**
	MPC-0.25-CCW	0.0657	0.6273	0.4830	3.0566
Complex shape	GVF-BCQ-0.4-CCW	**0.0341**	**0.2540**	**0.2272**	**1.5306**
	MPC-0.4-CCW	0.0625	0.5846	1.2133	8.1747
	GVF-BCQ-0.4-CW	**0.0348**	**0.2339**	**0.2240**	**1.3911**
	MPC-0.4-CW	0.0809	0.7521	1.2861	8.7905
	GVF-BCQ-0.25-CCW	**0.0006**	**0.0082**	**0.1696**	**1.0394**
	MPC-0.25-CCW	0.0525	0.4932	0.6457	3.6786



(a) Actor Model (b) VAE Model

Fig. 15: Neural network models for E2E-BCQ (a) Actor, and (b) VAE. The Critic model is the same as DDPG in Figure 21

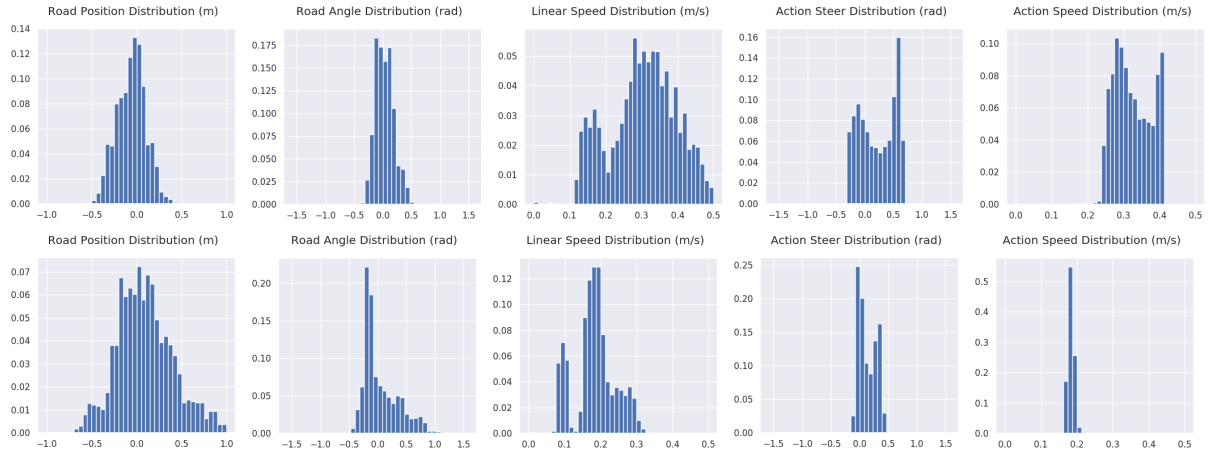


Fig. 16: Distribution of lane centeredness, road angle, speed and action distribution of GVF-BCQ and E2E-BCQ on the rectangle test track at 0.4 speed, counterclockwise direction.

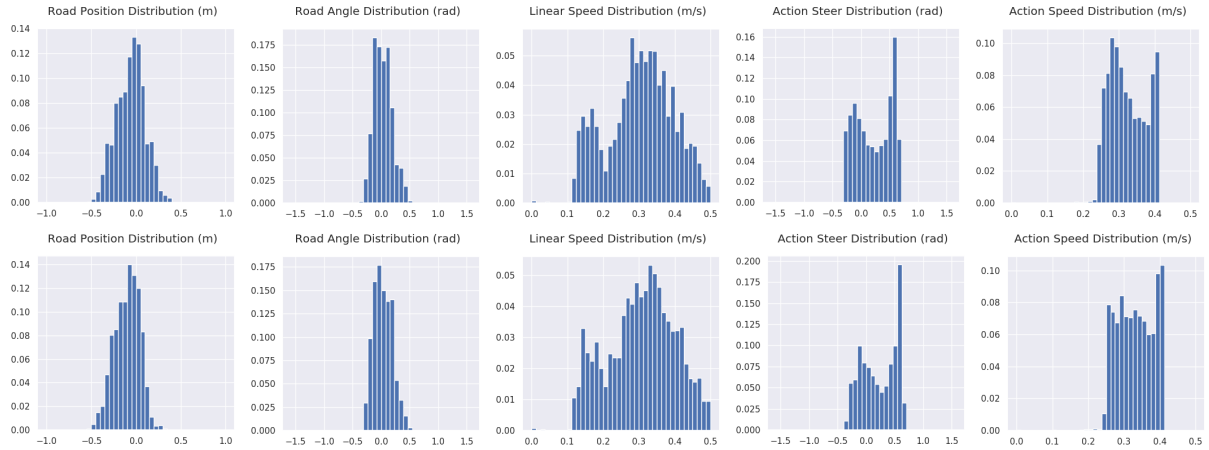


Fig. 17: Distribution of lane centeredness, road angle, speed and action distribution on the rectangle test road. From top to bottom: GVF-BCQ-0.4, GVF-BCQ-0.4 with lane marking damage on the rectangle road. The similarities highlight the robustness of GVF-BCQ to the introduction of damaged lanes.

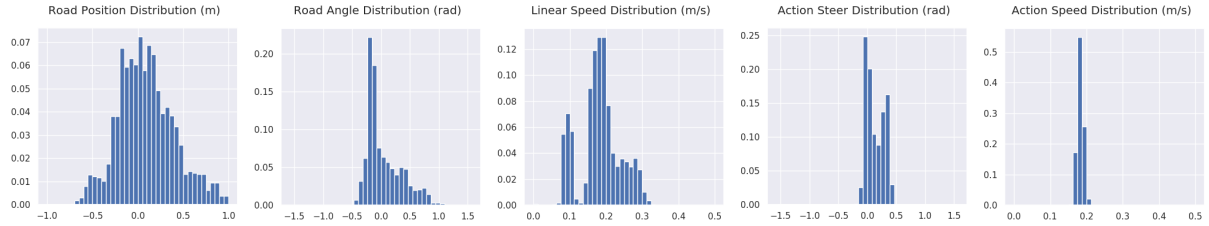


Fig. 18: Distribution of lane centeredness, road angle, speed and action distribution of E2E-BCQ on the rectangle test road at 0.4 speed, counterclockwise direction

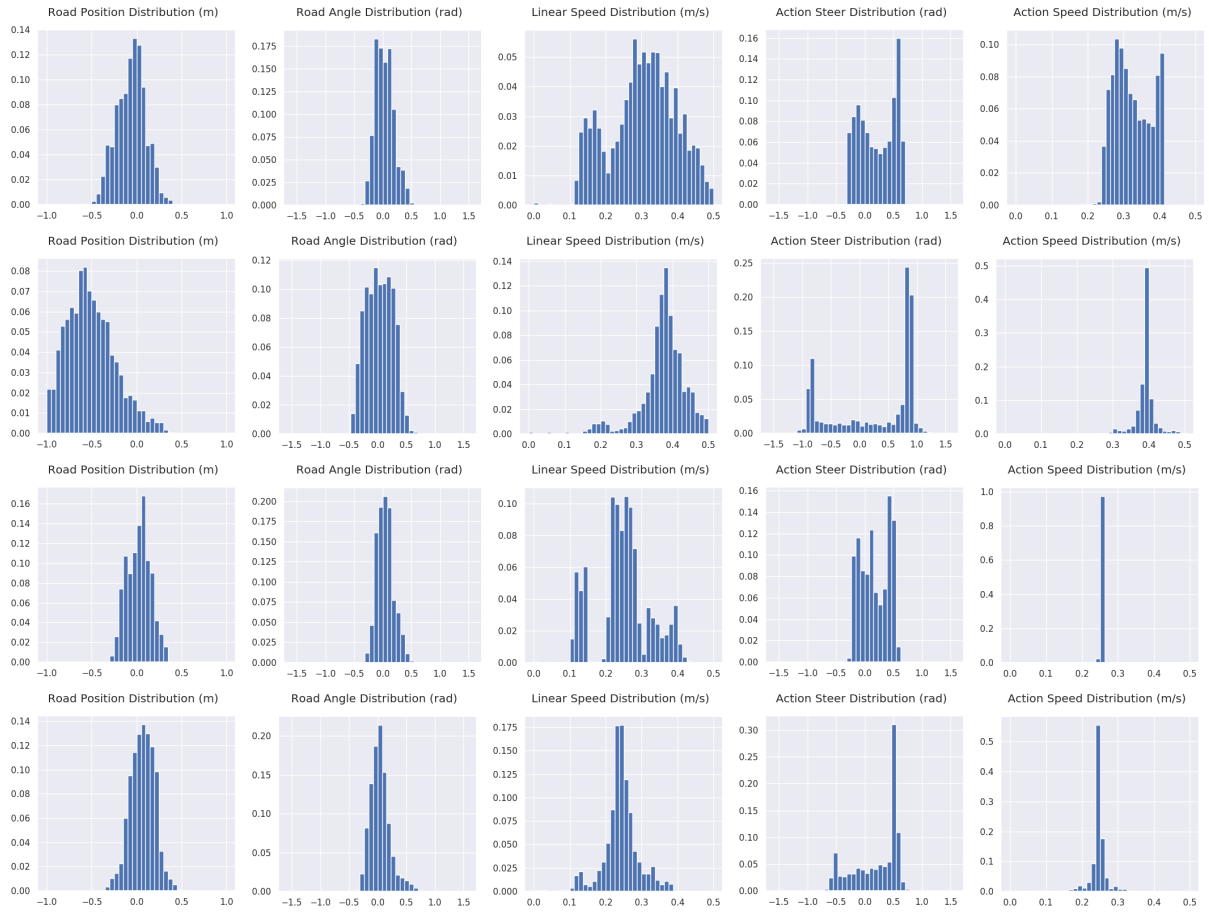


Fig. 19: Distribution of lane centeredness, road angle, speed and action distribution of GVF-BCQ and MPC at 0.4 m/s and 0.25 m/s on the rectangle test track. From top to bottom: GVF-BCQ-0.4-CCW, MPC-0.4-CCW, GVF-BCQ-0.25-CCW, MPC-0.4-CCW[7]

B. TORCS Experiments

TORCS is a racing simulator used for learning to drive. All opponent vehicles were removed for these experiments as well as roads that were sloped. The goal of the agent is to maximize the future accumulation of the following reward: $r_t = 0.0002 v_t(\cos\beta_t + |\alpha_t|)$ where v_t is the speed of the vehicle in km/h, β_t is the angle between the road direction and the vehicle direction, and α_t is the current lane centeredness. Termination occurs when either the agent leaves the lane or the maximum number of steps has been reached (1200 steps = 120 seconds) triggering a reset of the environment. Upon reset, a priority sampling method is used during training to select the next road to train on. The probability of sampling road i during a reset is given by

$$\frac{e^{-\frac{n_i}{\kappa}}}{\sum_{j=1}^{N} e^{-\frac{n_j}{\kappa}}} \tag{15}$$

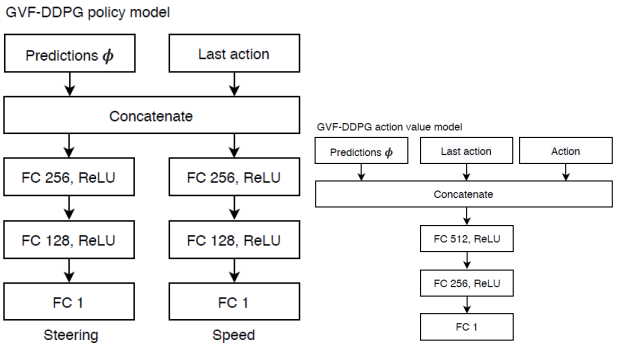
where n_i is the number of steps that the agent was able to achieve the last time the road was sampled and κ controls the spread of the distribution. A value of $\kappa = \frac{1}{N}\sum_{j=1}^{N} n_j$ was found to perform well. The initial probabilities are equal for all roads. This improved the efficiency in learning for all learned methods.

The TORCS environment was modified to provide higher resolution images in grayscale rather than RGB with most of the image above the horizon cropped out of the image. The grayscale images were 128 pixels wide by 64 pixels high. This allowed the agent to see more detail farther away which is very helpful in making long term predictions and is beneficial to both policy gradient methods and predictive learning.

1) Training: Two main learning algorithms are compared, along with their variants: our proposed GVF-DDPG (general value functions with deep deterministic policy gradient) and end-to-end DDPG. The parameters used to train the methods will be described in more detail here.

GVF-DDPG Training: Exploration followed the same approach as [57] where an Ornstein Uhlenbeck process [58] was used to explore the road; the parameters of the process ($\theta = 1.0$, $\sigma = 0.1$, $dt = 0.01$) were tuned to provide a gradual wandering behavior on the road without excessive oscillations in the action. This improved the learning of the off-policy predictions for GVF-DDPG since the behavior policy $\mu(a|s)$ was closer to the target policy $\tau(a|s)$ of the predictions.

The GVF-DDPG approach learned 8 predictions: 4 predictions of lane centeredness α, and 4 predictions of road angle β. Each of the 5 predictions had different values of γ for different temporal horizons: 0.0, 0.5, 0.9, 0.95, 0.97. This allowed the agent to predict how the road will turn in the future providing the necessary look ahead information for the agent to control effectively. The GVF predictors shared the same deep convolutional neural network as used on the Jackal robot where the convolutional layers were identical to the architecture in Figure 10 followed by three fully connected layers of 512, 384 and 8 outputs, respectively as shown in Figure 11a. The behavior estimator $\mu(a|s)$ was identical with the one used on the Jackal robot in Figure 11b. The models for actor and critic models for GVF-DDPG are given in Figures 20a and 20b. Actions produced by the actor network was clipped to the range $[-1.0, 1.0]$. A linear transformation was applied to the target speed action to change the range of values from $[-1.0, 1.0]$ to $[0.5, 1.0]$. The steering and target speed input into both policy and critic networks were normalized to range $[-1.0, 1.0]$.



(a) Actor Model (b) Critic Model

Fig. 20: Neural network models for GVF-DDPG (a) Actor, and (b) Critic

A replay buffer of size 100,000 was used with a warmup of 10,000 samples. In order to not bias the replay buffer, the last layers of the actor network were initialized with a uniform distribution $[-1e^{-3}, 1e^{-3}]$ for the weight and 0 for the bias. The learning rates for the actor network, critic network, predictor network and behavior policy network were $1e^{-6}$, $1e^{-4}$, $1e^{-4}$, and $1e^{-4}$ respectively. Target networks [57] were used for the critic and actor networks with $\tau = 0.001$ in order to make the bootstrapped prediction of the action-values more stable. However, target networks were not necessary for the GVF predictions or the behavior policy estimation. The reward for was scaled by 0.0002 to scale the action-values to fall within the range $[-1.0, 1.0]$.

Baseline DDPG Training: The two DDPG (deep deterministic policy gradient) [57] baselines were trained nearly identically where the only difference was the information provided in the observation. The first method called DDPG-Image is a vision-based approach where the image, current speed, and last action are provided to the agent; the only information available to the agent about the road is supplied via images. The second agent called DDPG-LowDim includes lane centeredness α and road angle β as part of the agent's state. The purpose was to understand the value of this information in learning when supplied as a cumulant in GVF-DDPG during training only or supplied an input to the actor and critic during training *and* testing. It should be noted that DDG-LowDim was the only learned approach that used α and β was inputs to the actor and critic networks during testing, whereas GVF-DDPG and DDPG-Image did not have access to this information during testing.

The network architectures for the DDPG actor and critic

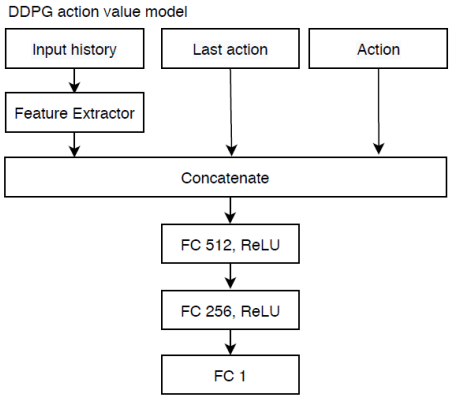


Fig. 21: Model of the DDPG critic network $Q(s, a)$.

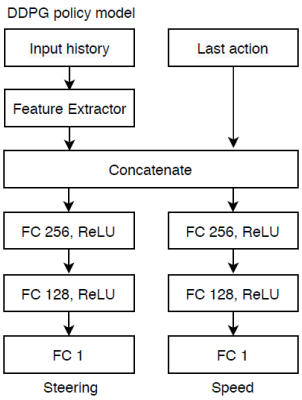
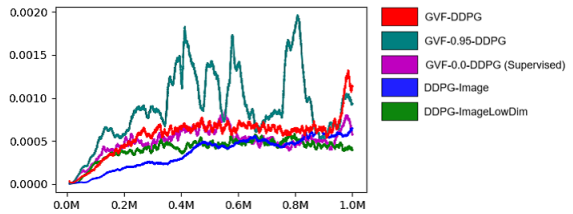


Fig. 22: Model of the DDPG actor network $\pi(s)$.

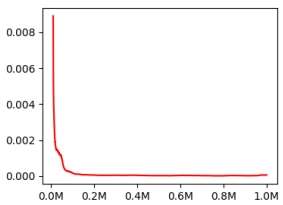
are given in Figures 22 and 21 respectively; they share the architecture for the feature extractor given in Figure 10.

The training setup for DDPG was the same as GVF-DDPG unless noted otherwise. An Ornstein Uhlenbeck process [58] was used to explore the road ($\theta = 1.0$, $\sigma = 0.1$, and $dt = 0.01$). Experimentally, it was found that the exploration parameters did not affect the learning performance of DDPG that much. Target networks were used with $\tau = 0.001$. The learning rates of the critic and actor networks were the same as those of GVF-DDPG.

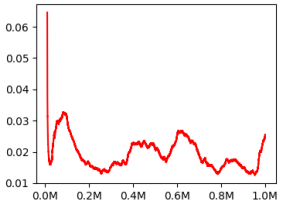
2) Experimental Results: The experimental results were averaged over 5 runs and mean and standard deviations plotted based on performance measured on the test roads during training. The learning curves for the critic networks for each of the DDPG agents, as well as learning curves for GVF predictions and behavior estimation for GVF-DDPG are shown in Figure 23. The average episode length is shown in Figure 24. The average lane centeredness and road angle during each episode are plotted in Figures 25 and 26. We can see how the GVF-DDPG with predictions over multiple time scales is able to maintain lane centeredness and road angle better than GVF-DDPG with myopic prediction ($\gamma = 0.0$) and future predictions with only $\gamma = 0.95$. The average lane centeredness and road angle is not substantially different though amoung the learned methods; however, DDPG-Image struggles a bit largely due to instability in learning as the high



(a) Critic



(b) Predictors



(c) Behavior

Fig. 23: Learning curves for (a) Q-values of the DDPG agents, (b) mean squared TD (temporal difference) errors of the GVF predictors, and (c) MSE of the behavior model estimator

variance is due to some failed runs where no learning occurs. Figure 27 shows the standard deviation in the change in the target speed action at each time step across an episode on each test road; this measures the jerkiness of the speed controller. GVF-DDPG and DDPG-LowDim are both able to control speed comfortably since the jerkiness is low. Finally, Figure 28 shows the lane centeredness on all six of the test roads during a final evaluation after training was completed. All six roads in the test set were challenging but the test roads a-speedway, alpine-2, and wheel-2 were especially challenging because the image of the roads were too different from the training roads. Nevertheless, on the dirt-4, evo-2-r, and spring roads, the lane centeredness of the the methods was quite good for all learned methods except for DDPG-Image. Note that while DDPG-LowDim performs well in learning to steer and control speed with lane centeredness and road angle, it required that information to learn to steer the vehicle which may be expensive or prohibitive to obtain in all situations such as in GPS-denied locations or locations where there is no map or it is out of date.

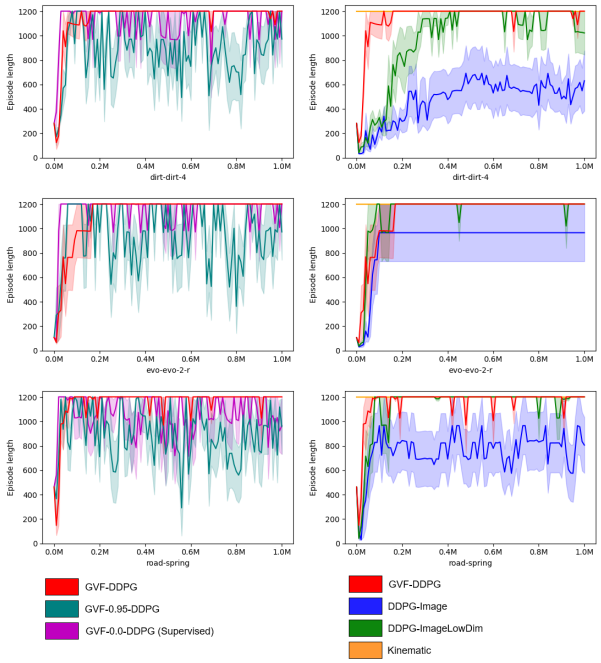


Fig. 24: Mean episode length during training for dirt-dirt-4, evo-evo-2 and road-spring

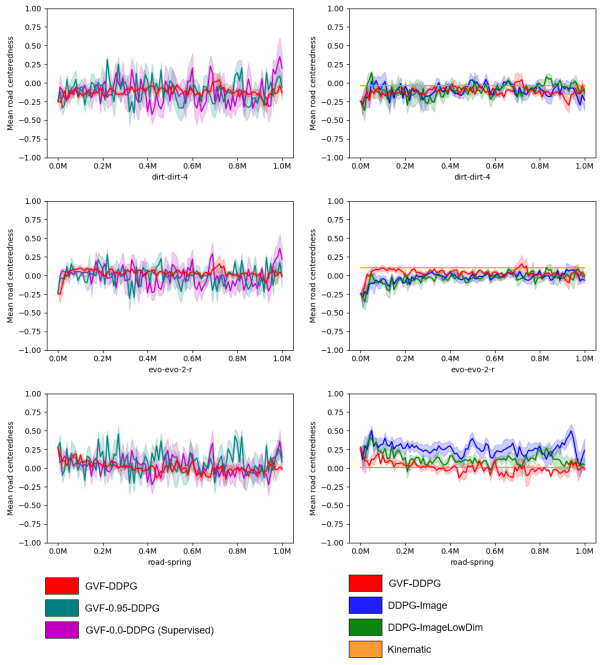


Fig. 25: Mean lane centeredness during training for dirt-dirt-4, evo-evo-2 and road-spring

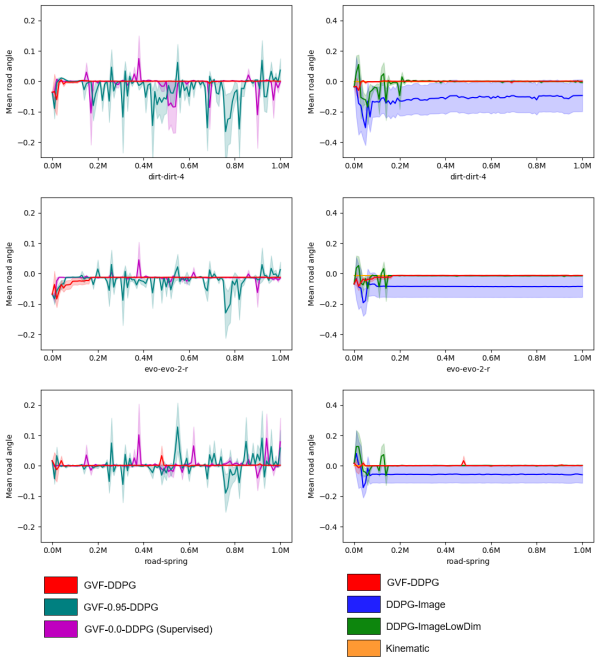


Fig. 26: Mean road angle during training for dirt-dirt-4, evo-evo-2 and road-spring

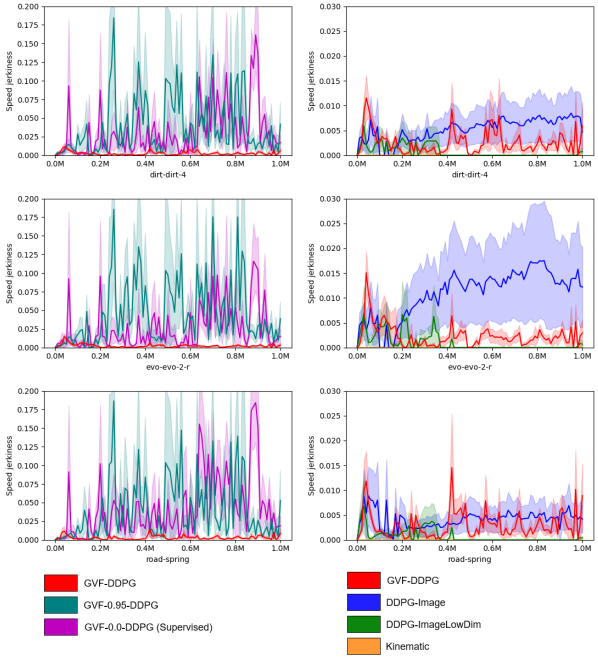


Fig. 27: Standard deviation of the change in target speed action during training for dirt-dirt-4, evo-evo-2 and road-spring

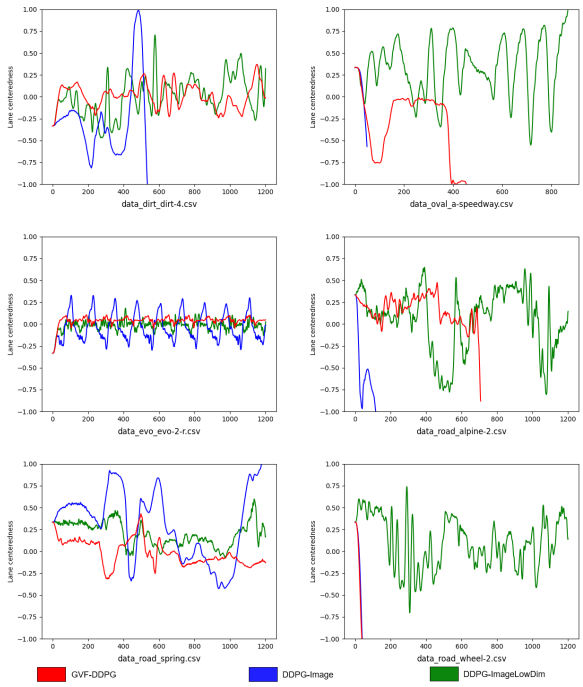


Fig. 28: The lane centeredness position on the (a) alpine-2, (b) evo-2-r, (c) dirt-4, (d) wheel-2, (e) spring, and (f) a-speedway roads in TORCS.

C. Predictive Learning Algorithms

The algorithm used for learning counterfactual predictions (GVFs) online through interaction with an environment is given in Algorithm 1. In important distinction of this algorithm is that the distribution of the behavior policy used to collect the data does not need to be known.

Algorithm 1 Online Counterfactual GVF training algorithm with unknown $\mu(a|s)$

Initialize $\phi^\tau(s)$, $g(a,s)$, $\eta(a|s)$, and replay memory D
Observe initial state s_0
for $t = 0,T$ **do**
    Sample action a_t from unknown $\mu(a_t|s_t)$
    Execute action a_t and observe state s_{t+1}
    Compute cumulant $c_{t+1} = c(s_t, a_t, s_{t+1})$
    Compute continuation $\gamma_{t+1} = \gamma(s_t, a_t, s_{t+1})$
    Estimate behavior density value $\hat{\mu}(a_t|s_t) = \frac{g(a_t,s_t)}{1-g(a_t,s_t)}\eta(a_t|s_t)$
    Estimate importance sampling ratio $\rho_t = \frac{\tau(a_t|s_t)}{\hat{\mu}(a_t|s_t)}$
    Store transition $(s_t, a_t, c_{t+1}, \gamma_{t+1}, s_{t+1}, \rho_t)$ in D
    Compute average importance sampling ratio in replay buffer D of size n with $\bar{\rho} = \frac{1}{n}\sum_{j=1}^{n}\rho_j$
    Sample random minibatch A of transitions $(s_i, a_i, c_{i+1}, \gamma_{i+1}, s_{i+1})$ from D according to probability $\frac{\rho_i}{\sum_{j=1}^{n}\rho_j}$
    Compute $y_i = c_{i+1} + \gamma_{i+1}\phi^\tau(s_{i+1};\hat{\theta})$ for minibatch A for most recent parameters $\hat{\theta}$
    Update parameters θ using gradient descent on (3) with gradient (6) over the minibatch A
    Sample random minibatch B of state action pairs (s_i, a_i) from D according to a uniform probability and assign label $z = 1$ to each pair
    Randomly select half the samples in the minibatch B replacing the action with $a_t \sim \eta(a|s)$ and label with $z = 0$ and storing the updated samples in $\hat{B}$
    Update behavior discriminator $g(a,s)$ with labels z in the modified minibatch $\bar{B}$ using binary cross-entropy loss

With minor modifications, an offline version of the algorithm can be derived. This algorithm learns by reading the data in sequence and populating a replay buffer just as it would in online learning; the only difference is that the offline algorithm returns the action taken in the data. This allows the same algorithm and code for learning counterfactual predictions (GVF) to be used in either online or offline learning settings.

Algorithm 2 Offline Counterfactual GVF training algorithm with unknown $\mu(a|s)$

Initialize $\phi^\tau(s)$, $g(a,s)$, $\eta(a|s)$, and replay memory D,
Obtain the first state in the data file s_0
for $t = 0,T$ **do**
    Obtain action a_t recorded in the data file that sampled from an unknown $\mu(a_t|s_t)$
    Obtain next state s_{t+1} from the data file
    Compute cumulant $c_{t+1} = c(s_t, a_t, s_{t+1})$
    Compute continuation $\gamma_{t+1} = \gamma(s_t, a_t, s_{t+1})$
    Estimate behavior density value $\hat{\mu}(a_t|s_t) = \frac{g(a_t,s_t)}{1-g(a_t,s_t)}\eta(a_t|s_t)$
    Estimate importance sampling ratio $\rho_t = \frac{\tau(a_t|s_t)}{\hat{\mu}(a_t|s_t)}$
    Store transition $(s_t, a_t, c_{t+1}, \gamma_{t+1}, s_{t+1}, \rho_t)$ in D
    Compute average importance sampling ratio in replay buffer D of size n with $\bar{\rho} = \frac{1}{n}\sum_{j=1}^{n}\rho_j$
    Sample random minibatch A of transitions $(s_i, a_i, c_{i+1}, \gamma_{i+1}, s_{i+1})$ from D according to probability $\frac{\rho_i}{\sum_{j=1}^{n}\rho_j}$
    Compute $y_i = c_{i+1} + \gamma_{i+1}\phi^\tau(s_{i+1};\hat{\theta})$ for minibatch A for most recent parameters $\hat{\theta}$
    Update parameters θ using gradient descent on (3) with gradient (6) over the minibatch A
    Sample random minibatch B of state action pairs (s_i, a_i) from D according to a uniform probability and assign label $z = 1$ to each pair
    Randomly select half the samples in the minibatch B replacing the action with $a_t \sim \eta(a|s)$ and label with $z = 0$ and storing the updated samples in $\hat{B}$
    Update behavior discriminator $g(a,s)$ with labels z in the modified minibatch $\bar{B}$ using binary cross-entropy loss