International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878, Volume-7, Issue-6, March 2019

Game Environment Exploration Using Curiosity-Driven Learning

Harihara Sudhan N, Shriram S, Meghna Anand, Sujeetha R

Abstract: Reinforcement learning (RL) has emerged as a preferred methodology for coaching agents to perform complicated tasks. In several real-world situations, rewards extrinsic to the agent are very distributed or absent altogether. In such cases to change the agent learn new skills and explore its surroundings, the curiosity will act as an intrinsic reward signal which may be helpful later in its life. The concept of Curiosity-Driven learning is to make a rewarding work that is characteristic for the agent (produced by the operator itself). It implies the operator will be a self-student since he will be the understudy here. However additionally the feedback master. An agent learns quickly if every of its action incorporates a reward, so he gets swift feedback. Curiosity is an intrinsic reward that's equal to the error of our agent to predict the consequence of its own actions given its current state (aka to predict subsequent state given current state and action taken). We demonstrate our output in a 3D simulated virtual environment.

Index Terms: Curiosity learning, Feedback, Rewards, Reinforced Learning, Virtual Environment.

I. INTRODUCTION

Newer companies with self-driving car investments such as Tesla, Google, Delphi, Nvidia, Mobileye, Nutonomy, Otto and Cruise Automation, most of whom have emerged from the vicinity of Silicon Valley, would trace the histories of their transport work back to the 'Grand Challenge' robot driving competitions staged by the Defense Advanced Research Projects Agency (DARPA). At the first event, in the Mojave Desert in 2004, the best of the cars managed only 7 miles of the 150-mile course. The next year, five vehicles completed the course. The 2007 version was staged in an urban environment. The six cars that finished were seen as an announcement to the world that innovation was happening apace. The 2005 and 2007events brought attention to engineers from Stanford and Carnegie Mellon universities, who would go on to populate some of the leading private-sector self-driving car teams. In this paper, we will explore the ways to employ curiosity-based learning models and use it to explore the 3D virtual environment. The job of curiosity has been generally contemplated in the specific circumstance of settling assignments with inadequate prizes. As we would like to think, curiosity has two other key employment. Curiosity helps an agent investigate its condition in the journey for new learning (an attractive normal for exploratory conduct is that it ought to improve as the operator acquires learning). Further, curiosity is a component for an agent to learn abilities that may be useful in future situations. In this paper, we assess the adequacy of our curiosity plan on the whole three of these jobs. We will try to simulate a self-driving car using this procedure. This work has a place with the general classification of strategies that produce an intrinsic reward signal dependent on how hard it is for the operator to anticipate the outcomes of its own behavior, for example, predict the following state given the present state and the executed activity. Be that as it may, we figure out how to escape generally traps of past expectation approaches with the accompanying key understanding: we just foresee those adjustments in nature that could be because of the activities of our operator or influence the operator, and overlook the rest. That is, rather than making forecasts in the raw sensory space (for example pixels), we change the tactile contribution to an element space where just the data applicable to the activity performed by the agent is spoken to. We get familiar with this element space utilizing self-supervision – preparing a neural network on an intermediary converse elements errand of anticipating the agent's activity given its present and next states. Since the neural network is as it was required to foresee the activity, it has no motivating force to speak to inside its component installing space the elements of variety in the condition that don't influence the operator itself. We at that point utilize this component space to prepare a forward elements display that predicts the component portrayal of the following: state, given the element portrayal of the present state also, the activity. We give the forecast blunder of the forward elements model to the operator as an intrinsic reward to support its interest. We will try to run a self-driving car that explores the 3D environment using the aforementioned method.

II. RELATED WORKS

Ongoing advancement in PC vision has been driven by high-limit models prepared on extensive datasets. Tragically, making huge datasets with pixel-level labels has been very exorbitant because of the measure of human exertion required. In this paper, we present a way to deal with quickly making pixel-exact semantic name maps for pictures separated from current PC diversions. Since the principal rivalry held in 2012, unique methodologies are utilized by

Revised Manuscript Received on March 25, 2019.

HariHara Sudhan N, Student, Department of CSE, SRM Institute of Science and Technology, Chennai, Tamil Nadu , India

Shriram S, Student, Department of CSE, SRM Institute of Science and Technology, Chennai, Tamil Nadu , India

Meghna Anand, Student, Department of CSE, SRM Institute of Science and Technology, Chennai, Tamil Nadu , India

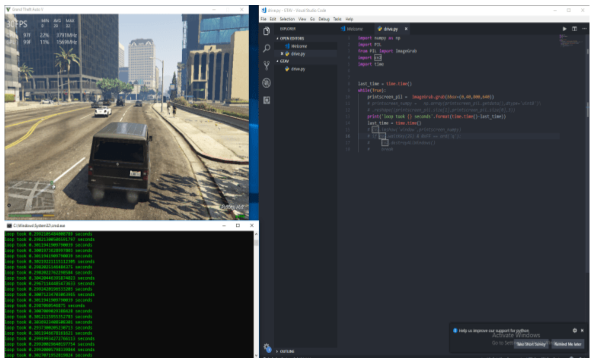
Sujeeta R, Assistant Professor ,Department of CSE, SRM Institute of Science and Technology, Chennai, Tamil Nadu ,India
.

members, for example, logical programming, subjective thinking, propelled reproduction, auxiliary examination, investigation of anticipated harm, and diverse AI procedures, for example, Bayesian outfit learning, Weigh greater part calculation, Heuristic hunt are connected. Be that as it may, to the best of our insight, there hasn't any group who included profound fortification figuring out how to tackle this issue. Furthermore, that is what our work was for the most part about. We approve the existing methodology by delivering thick pixel-level semantic explanations for 25 thousand pictures incorporated by a photorealistic open-world PC amusement. Investigations on semantic division datasets demonstrate that utilizing the obtained information to enhance certifiable pictures altogether builds exactness and that the gained information empowers lessening the measure of hand-marked genuine information: models prepared with amusement information and only 1/3 of the CamVid preparing set beat models prepared on the total CamVid preparing set.

Paul and Hllermeier [Paul and Hullermeier, 2015¨] constructed a case-based angry birds agent for 2D. They utilize an inclination based approach which looks at changed answers for guaranteed issue and keeps up the better one to fabricate the case base, and after that attempt to locate the most comparative diversion scene from its case base and receives its shot when playing another amusement[1]. Calimeri, Fink and Germano [Calimeri et al., 2016] expand Answer Set Programming with outside sources and different augmentations dependent on explanatory information bases. They rank each shootable target thinking about their probability to devastate other objects for a solitary shoot[2]. Polceanu and Buche [Polceanu and Buche, 2014] form their basic leadership dependent on the hypothesis of mental recreation. Their operator watches the impacts of playing out various reproductions of various shots in a given diversion scene and chooses the ideal arrangement dependent on these outcomes[3].

III. PROPOSED SYSTEM

Today, there are two noteworthy ideal models for vision-based autonomous driving systems: interceded discernment approaches that parse a whole scene to settle on a driving choice, and conduct reflex methodologies that specifically map an information picture to a driving activity by a regressor[4]. In this paper, we propose a third worldview: an immediate discernment approach to gauge the affordance for driving. We chose 3D environment because we thought it would be challenging for the agent to train and work as well as . *The game gets to familiarize itself with the reward factors*. RL needs a reward function. We can't just peruse out progress criteria from the procedure memory, as there is an excess of variety in how each diversion stores this data. Luckily, numerous diversions have an on-screen score which we can use as a rewarding work, as long as we can parse it. While off-the-rack OCR, for example, Tesseract performs incredibly on standard text styles with clean foundations, it battles with the differing textual styles, moving foundations, showy movements, or blocking objects normal in numerous amusements. We built up a convolutional neural system based OCR show that keeps training the player model until it can navigate from one point to another. During the first few hours of training, the player model back and forth optimizing it's reward values. After 3-4 hours the car could navigate but not optimally. After 10 hours of training, the car could navigate through the streets of a fictional depiction of Los Santos. We built another layer of CNN to help assist the agent to improve its navigation. First, we designed screen capture using OpenCV and Python. This was to grab whatever was on the screen. For this, we took the dimensions to match a 800x600 resolution of GTA V in windowed mode.


drive.py - GTAV - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
EXPLORER
OPEN EDITORS
Welcome
drive.py
GTAV
drive.py

```
import numpy as np
import PIL
from PIL import ImageGrab
import cv2
import time


last_time = time.time()
while(True):
    printscreen_pil =  ImageGrab.grab(bbox=(0,40,800,640))
    # printscreen_numpy =   np.array(printscreen_pil.getdata(),dtype='uint8')\
    # .reshape((printscreen_pil.size[1],printscreen_pil.size[0],3))
    print('loop took {} seconds'.format(time.time()-last_time))
    last_time = time.time()
    # cv2.imshow('window',printscreen_numpy)
    # if cv2.waitKey(25) & 0xFF == ord('q'):
    #     cv2.destroyAllWindows()
    #     break
```

Fig.1. Capturing frames from the game.

Then we configured the keybinds as per requirement with the basic being W, A, S, D to be directional movements. To simulate pressing keys in python, we used to scan codes for the following keys.

W = 0x11

A = 0x1E

S = 0x1F

D = 0x20

Now that we can read frames and simulate keystrokes, we move forward to detection of lanes on the road. We can lane detection to both make a self-driving AI that works dependent on basic guidelines dependent on these paths and furthermore to prepare an AI that we expect could later sum up to more situations.

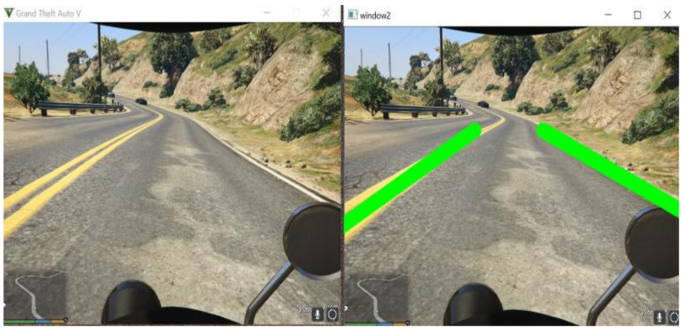
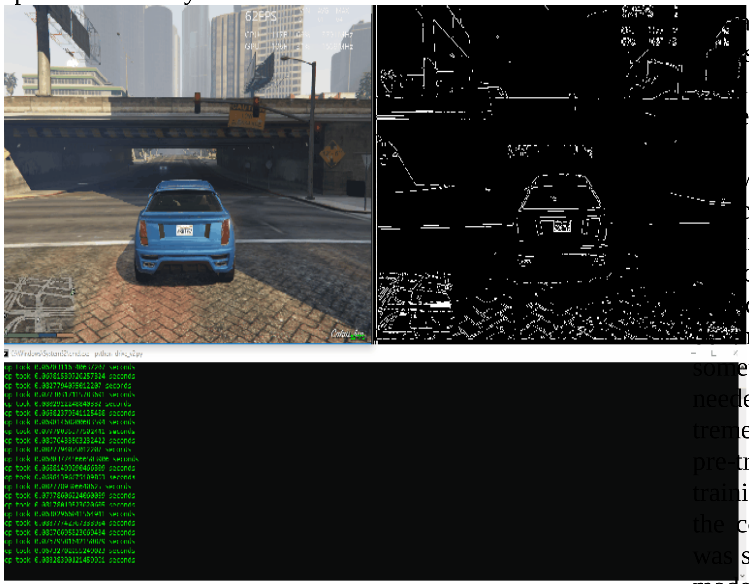


Fig.2. Result of Lane Detection algorithm.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

Next, we add the Hough Line algorithm to detect lines. After adding a bit of Gaussian blur to counter the aliasing, we were able to establish detection of lines(lanes). The longest and thickest line will be our lanes. Now, the goal is to determine from these lines, which are likely our actual lanes. Next, take the two most normal lines, and expect these to be our paths. After we've done ROI, the following undoubtedly "line" basically is practically sure to be the paths. That is the speculation in any case.



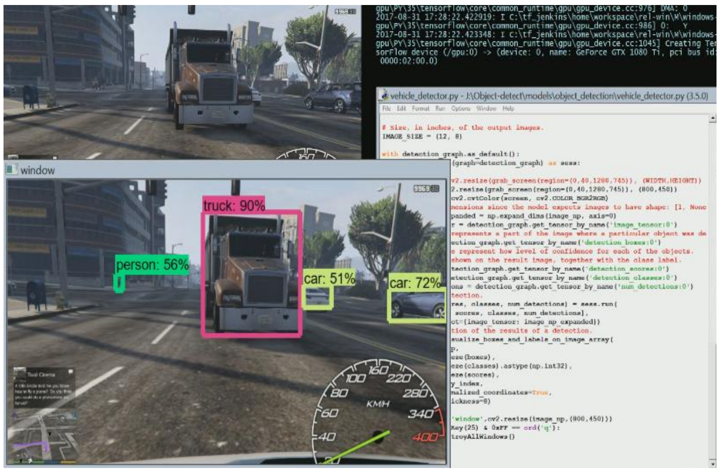
```
C:\Windows\System32\cmd.exe - python drive_v2.py
op took 0.0[illegible] seconds
op took 0.00781580720257324 seconds
op took 0.0827704075612207 seconds
op took 0.0[illegible] seconds
op took 0.0882[illegible] seconds
op took 0.0682370341125438 seconds
op took 0.0[illegible] seconds
op took 0.0[illegible] seconds
op took 0.0837043503232422 seconds
op took 0.0[illegible] seconds
op took 0.0[illegible] seconds
op took 0.0881400206460380 seconds
op took 0.0[illegible] seconds
op took 0.0[illegible] seconds
op took 0.0777600224660880 seconds
op took 0.0[illegible] seconds
op took 0.0[illegible] seconds
op took 0.0[illegible] seconds
op took 0.0837600552360484 seconds
op took 0.0[illegible] seconds
op took 0.0[illegible] seconds
op took 0.0832830121450001 seconds
```

Fig.3. Canny edge detection algorithm application.

Now we finally go into incorporating our Artificial Intelligence system. The logic for driving works as follows: Log the slopes of the lane

- When the one path's slope is certain, and the other is negative, proceed with straight.
- When both path's slopes are negative, at that point we're excessively far left, turn right.
- When both path's inclines are certain, at that point we're excessively far right, turn left.

The arrangement so far has been to initially think of some fundamental driving principles, and ideally recognize regardless of whether we were between two paths. This will be a neural network system, it will take a few sources of info, and produce a yield. Our information will be screen information. We could send in simply the ROI we were utilizing previously, yet we considered simply sending in the whole screen. The yield is real driving directions. As expressed above, it is perfect to have a PID framework, in any case, for the present, we have the "win big or bust" setup. In this way, fundamentally the contribution to the system is screen information and the yield will be to press the W key or the A key, etc. We need highlights and marks! To do this, we have to record ourselves playing GTA V, sparing the screen information as what will be neural system input, the pixels as highlights, and after that our keypresses, which will be the yield target layer when preparing the system. For training data, we need two features and labels. The capabilities will be the pixel data from the screen data. The labels are the key information sources we need the AI to make. We as of now have the pixel data, however, we don't have a method for gathering inputs. We made data sources, yet we need an approach to catch our own information sources. Likewise Eventually, in any case, we truly would require a database or something to that effect on the off chance that we needed to have an immaculate AI, since it would require gigantic measures of training data. In the end, be that as it may, we truly would require a database or something to that effect when the situation arises where we needed to have an immaculate AI, since it would require tremendous measures aining data. For training data, we need two features and s. The capabilities will be the pixel data from the screen. abels are the key information sources we need the AI to . We as of now have the pixel data, however, we don't a method for gathering inputs. We made data sources, we need an approach to catch our own information es. Likewise eventually, in any case, we truly would re a database or something to that effect on the off ce that we needed to have an immaculate AI, since it d require gigantic measures of training data. In the end, at as it may, we truly would require a database or something to that effect when the situation arises where we needed to have an immaculate AI, since it would require tremendous measures of training data. We will be using pre-trained data from Deep Drive[5]. This will save us a lot of training time. Now we're ready to see how we've done. While the convolutional neural network was trained, the progress was saved on a model file. This lets us easily track back the model and alternately use it or train it more. Now we go with the final phase of the implementation which is object detection and collision warning system with TensorFlow APIs that tracks the cars and other nearby objects and mentions whether a collision is imminent. If a car is beyond a certain threshold value we set, the system raises and alarm notifying the user that a collision is about to occur. This value will depend on number of factors like The density of the traffic, the reliability of the agent, the velocity of the car, the behavior of AI in NPCs and so on. This is one of the system which we believe will need a lot of training and tweaking in the future to polish the accuracy and reduce errors. Our simulation environment is now ready.



```
gpu\PY\35\tensorflow\core\common_runtime\gpu\gpu_device.cc:976] DMA: 0
2017-08-31 17:28:22.422919: I C:\tf_jenkins\home\workspace\rel-win\M\windows-
gpu\PY\35\tensorflow\core\common_runtime\gpu\gpu_device.cc:986] 0:   Y
2017-08-31 17:28:22.423348: I C:\tf_jenkins\home\workspace\rel-win\M\windows-
gpu\PY\35\tensorflow\core\common_runtime\gpu\gpu_device.cc:1045] Creating Ten
sorFlow device (/gpu:0) -> (device: 0, name: GeForce GTX 1080 Ti, pci bus id:
 0000:02:00.0)
```

```
vehicle_detector.py - J:\Object-detect\models\object_detection\vehicle_detector.py (3.5.0)
File  Edit  Format  Run  Options  Window  Help

# Size, in inches, of the output images.
IMAGE_SIZE = (12, 8)

with detection_graph.as_default():
        (graph=detection_graph) as sess:

v2.resize(grab_screen(region=(0,40,1280,745)), (WIDTH,HEIGHT))
2.resize(grab_screen(region=(0,40,1280,745)), (800,450))
cv2.cvtColor(screen, cv2.COLOR_BGR2RGB)
mensions since the model expects images to have shape: [1, None
panded = np.expand_dims(image_np, axis=0)
r = detection_graph.get_tensor_by_name('image_tensor:0')
represents a part of the image where a particular object was de
ection_graph.get_tensor_by_name('detection_boxes:0')
e represent how level of confidence for each of the objects.
shown on the result image, together with the class label.
tection_graph.get_tensor_by_name('detection_scores:0')
etection_graph.get_tensor_by_name('detection_classes:0')
ons = detection_graph.get_tensor_by_name('num_detections:0')
tection.
res, classes, num_detections] = sess.run(
 scores, classes, num_detections],
ct={image_tensor: image_np_expanded})
tion of the results of a detection.
sualize_boxes_and_labels_on_image_array(
p,
eze(boxes),
eze(classes).astype(np.int32),
eze(scores),
y_index,
malized_coordinates=True,
ickness=8)

'window',cv2.resize(image_np,(800,450)))
Key(25) & 0xFF == ord('q'):
troyAllWindows()
```

Fig.4. Collision Detection.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

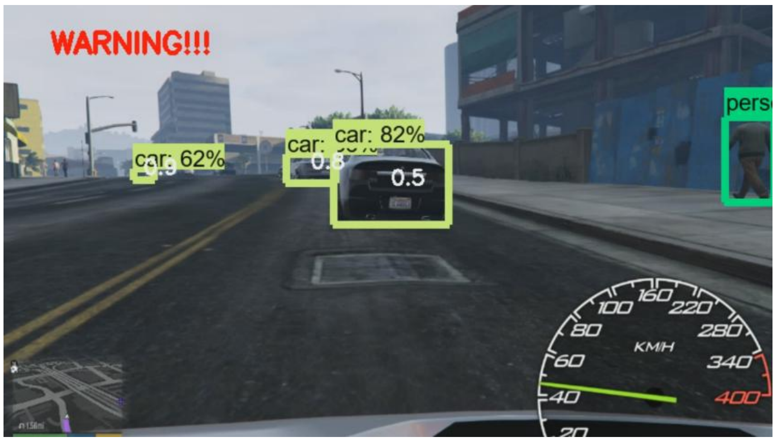


Fig.5. Collision Warning.

IV. RESULT

Our entire continuous pipeline including lane detection, navigation, and collision detection are completely functional in our model. The AI agent seems to drive. Since the NPCs are controlled by another AI, the collision system's accuracy may need some tweaking. The collision warning system deploys an error message right before we are about to collide. We were able to follow a relative separation to vehicles around us, and decide whether they're excessively close and flag a notice and additionally even issue some evasive controls with recreated keypresses. In GTA V explicitly, you need direct input sources. We trained over 25000 samples and the agent seems to function pretty decently. We're pretty pleased with our results.

REFERENCES

1. Paul and Hullermeier, 2015 ¨] Adil Paul and Eyke Hullermeier. A cbr approach to the angry birds game. In ¨ ICCBR (Workshops), pages 68–77, 2015.
2. Calimeri et al., 2016] Francesco Calimeri, Michael Fink, Stefano Germano, Andreas Humenberger, Giovambattista Ianni, Christoph Redl, Daria Stepanova, Andrea Tucci, and Anton Wimmer. Angry-hex: an artificial player for angry birds based on declarative knowledge bases. IEEE Transactions on Computational Intelligence and AI in Games, 8(2):128–139, 2016.
3. Polceanu and Buche, 2014] Mihai Polceanu and Cedric Buche. Towards a theory-of-mind-inspired ´ generic decision-making framework. arXiv preprint arXiv:1405.5048, 2014
4. DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving, Chenyi Chen, Ari Seff,,Alain Kornhauser Jianxiong Xiao.
5. Deep Drivc - https://deepdrive.io/.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication