# GTAV Lane Marking Detection

Aitor Ruano (aruanom@uoc.edu)
Open University of Catalonia

## Introduction

Detecting lane markings is a mandatory requirement for the correct execution of vision-based self-driving agents, either to compute the lines trajectory, detect lanes, or estimate the distance of the vehicle to the center of the road[1].

In open-source games such as Torcs[2], that have been widely used by researchers, information about the road lines is explicitly available. However, in Grand Theft Auto V this is not the case, not even by using the native function calls provided by the modding community via ScriptHookV[3].

Obtaining real-time information about the lines in GTAV is key for deep learning researchers to adopt this much more realistic environment, either to be used for supervised or reinforcement learning. This document serves as a guide to explain how this information can be accessed and all the code is freely available in my GitHub repo.

## Nodes and Links

In GTAV the map paths are defined by nodes and links that are usually used by AI driven characters or vehicles to follow a route and travelling according to “traffic regulations”, not only on asphalted roads but also on narrow secondary paths and water[4].

Nodes are 3D points with associated attributes, and are always located in the center of the road, to better approximate the trajectory in curves, the density of nodes is higher in them. Links in the other hand, are unions between pairs of nodes, also with their associated attributes.

Access to the nodes can be obtained by using native calls[5] such as:

```
BOOL GET_NTH_CLOSEST_VEHICLE_NODE(float x, float y, float z, int nthClosest, Vector3
*outPosition, Any unknown1, Any unknown2, Any unknown3)
```

Unfortunately, it seems that no information about the links is available through in-game function calls.

---

[1] http://deepdriving.cs.princeton.edu/
[2] http://torcs.sourceforge.net/
[3] http://www.dev-c.com/gtav/scripthookv/
[4] Doesn’t seem to exist for flight routes.
[5] http://www.dev-c.com/nativedb/

Until recently the meaning of the attributes for nodes and links where unknown as indicated by the GTA Wiki[6]. However by accessing the GTA V files with OpenIV[7] one can obtain the *paths.xml* file, which includes all the information from all nodes and links of the game (77248 nodes and 80592 links), even for the meaning of the attributes. This file can be downloaded from this link.

Here follows a description of the attributes:

```
//Between ( ) default values

typedef struct {
    bool disabled; //Disabled (0)
    bool water; //Water (0)
    int speed; //Speed (4)
    int special; //Special (0)
    int density; //Density (-1)
    bool highway; //Highway (0)
    bool noGPS; //NoGps (0)
    bool tunnel; //Tunnel (0)
    bool cantGoLeft; //Cannot Go Left (0)
    bool leftTurnsOnly; //Left Turns Only (0)
    bool offRoad; //Off Road (0)
    bool cantGoRight; //Cannot Go Right (0)
    bool noBigVehicles; //No Big Vehicles (0)
    bool indicateKeepLeft; //Indicate Keep Left (0)
    bool indicateKeepRight; //Indicate Keep Right (0)
    bool slipLane; //Slip Lane (0)
} tNodeAttr;

typedef struct {
    int width; //Width (0)
    int lanesIn; //Lanes In (1)
    int lanesOut; //Lanes Out (1)
    bool narrowRoad; //Narrowroad (0)
    bool gpsBothWays; //GpsBothWays (0)
    bool blockIfNoLanes; //Block If No Lanes (0)
    bool shortcut; //Shortcut (0)
    bool dontUseForNavigation; //Dont Use For Navigation (0)
} tLinkAttr;



typedef struct{
    Vector3 coord;
    Vector3 direction;
    tLinkAttr attr;
    std::string ref1;
    std::string ref2;
} tLink;
```

[6] http://gta.wikia.com/wiki/Main_Page
[7] http://openiv.com/

```
typedef struct{
    int id;
    Vector3 coord;
    tNodeAttr attr;
    std::vector<tLink> links;
} tNode;

typedef std::unordered_map<int, tNode> NodesMap;
```

It is easy to observe that most of the relevant information to estimate the position of the lane markings is encoded in the coordinates and attributes: *highway, narrowRoad, lanesIn, lanesOut and width.*

This information stored in paths.xml can be loaded into RAM for real-time faster access during the game. GTAV generates a unique integer ID for each node that can be accessed by the following native call:

```
int GET_NTH_CLOSEST_VEHICLE_NODE_ID(float x, float y, float z, int nth, int nodetype, float
p5, float p6)
```

Where *nodetype* is 0 to retrieve asphalt road nodes only, 1 for all roads and 3 for water nodes. The last two arguments of the function are unknown, but from experimentation it seems that are ignored. *x*, *y* and *z* are the position coordinates from which you want to obtain the *nth* closest node id.

This is the fastest way to know the surrounding closest nodes to your current location. Unfortunately, there is no field for the node ID in *paths.xml* so there is no way to directly associate the attributes obtained from the file to the node given by the native call.

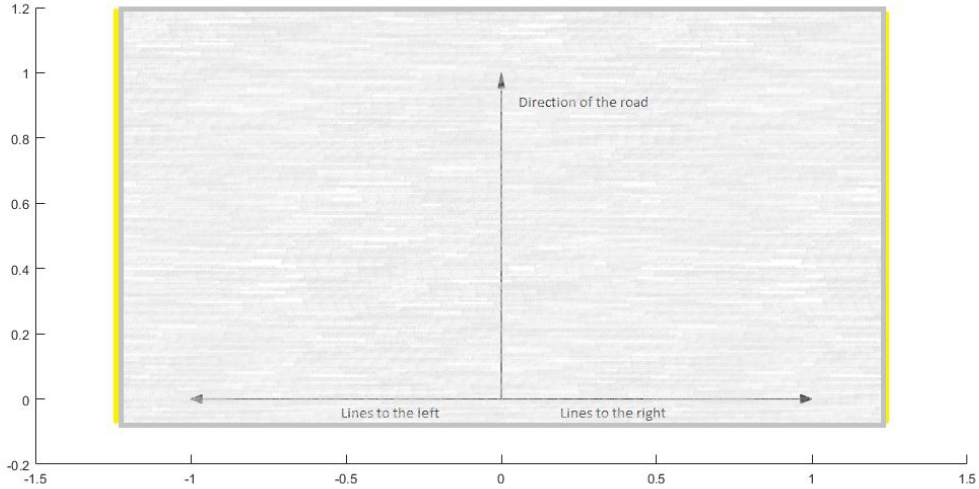
This is easily solvable by using the same function call with the coordinates of the nodes read in the file. The closest node to the position of the node in the file shall be the same node. Then, this integer ID can be used as key for a map of nodes, for easy access to all the nodes information. Finally, *paths.xml* file stores also the relation of the links to each pair of nodes, so this can also be looked up to store the relevant link information for each node.

The function that stores all the road paths of GTAV in a map can be found in *nodes.cpp*.

## Estimating the position of the lines

Each link does not only reference a pair of nodes but it also has an intrinsic direction that goes from node 2 to node 1. In the majority of cases, this direction follows the road, in others however, such as in shortcut nodes, this is not true and they should be ignored. We can take

this into account to create a vector pointing from node 2 to node 1 (parallel to the road) for every link, and then use it to draw two orthogonal vectors pointing to the lane lines, as the lines always follow the road (or they should!).




After some experimentation, it was easy to observe that the *lanesIn* attribute is the number of lanes to the “right” of the vector in the direction of the link, while *lanesOut* is the number of lanes to the “left”.

What should be the norm of the line vectors? Well, it depends on the lane width, number of lanes and the type of road. The attribute *width* is an integer that goes from -1 to 5 indicating a predefined length of the road, so a table had to be made relating an array of relevant attributes to the “real width” of the GTAV roads, measuring them from the outer lines. Here follows some of the measured values:

| **Narrow Road** | **Highway** | **Width** | **Real Lane Width** |
|---|---|---|---|
| 1 | 1 | 0 | 4.2 |
| 0 | 0 | 0 | 5.5 |
| 0 | 0 | 1 | 6.0 |
| 0 | 0 | -1 | 3.5 |
| 0 | 0 | 5 | 6.0 |

Finally the number of lines for each side of the road (right or left) is easily calculable with the following equation:

$$floor(1 + \frac{lanesIn + lanesOut}{2})$$

Note that if the result is even, it means that the node is already on one of the lines (the center line). This is because the center line is counted twice, one for each side of the road.

The code to calculate the line locations can be found in *nodes.cpp*.

Once the line points are known, it is pretty straightforward to calculate the distance of the vehicle to each one of them by computing the projection of the vehicle to the vector from the node to the line, as shown in the following piece of code:

```
float a = GAMEPLAY::GET_ANGLE_BETWEEN_2D_VECTORS(vehicle.coord.x - linePoint.x,
vehicle.coord.y - linePoint.y, v.x, v.y);
float d = GAMEPLAY::GET_DISTANCE_BETWEEN_COORDS(vehicle.coord.x, vehicle.coord.y, 0,
linePoint.x, linePoint.y, 0, FALSE)*SYSTEM::COS(a);
```

## Conclusions & next steps

This is an effective and somewhat accurate method to detect the GTAV lane markings, using only data provided by the game. It is also an important feat in order to give GTAV enough features to be used by Deep Learning researchers, information about the lane markings was essential to be able to compute supervised learning labels or reinforcement learning rewards.

There are still some roads to solve, most of them located in urban scenarios, where lanes are not so well defined and more insight is needed, especially because more attributes are involved and things get more complex, but this is just a matter of time and patience.