Abstract

Master of Electrical Engineering Program

Cornell University

Design Project Reports

Project I Title:

DARPA Urban Challenge, a C++ based platform for testing Path Planning Algorithms

Author: Raphael Rubin

Abstract:

The DARPA Grand Challenge in which the Cornell Racing Team participates requires the completion of a Simulator, which purports all errors in the artificial intelligence path planning down below and back up.

The simulator comes as the last layer in the top down approach followed by the Cornell Racing Team.

The Strategic layer is charged of global route planning, the tactical layer of collision avoidance and maneuver planning, while the operational layer controls lane tracking and safe following.

The simulator is the last layer. Through a COBRA interface the C++ or C# version of the simulator will be receiving commands from the Artificial Intelligence Strategic Layer concerning maneuvers such as Turn Left, Turn Right, Change Lane, Increase Speed, and Stop.

The simulator induces from its current situation, using controls such as bounding boxes and the World class, pointing to every object in the World, a set of more detailed commands.

Apart from writing a simplified version of the simulator in C++, we also concentrated my efforts onto finding a solution aside from dynamic programming for Path Planning and the Behavioral Modeling of Visible and Neighboring Vehicles on the road network.

We have built an efficient and self-correcting C++ GUI Interface including some random moving vehicles as well as a smart vehicle named Autosmart.

The Path Planning algorithm is written and implemented although may be missing a more significant round of testing.

To do so, we are using the approach of game theory and artificial intelligence's neural networks. We represent the world as nature, resulting in decisions independent of the drivers (types: turn left or right at the next intersection); nature being in this case the DARPA Challenge organizers. Moreover the drivers chose their behaviors (aggressive, altruist) on the road and keep updating their anticipations about the other players behavior and types, as mentioned above.

The end result is to train these neural networks to react to previously categorized behaviors and situations by storing necessary information about the 'game'.

Every player runs its own network, although in our case we limited the simulation to one smart vehicle, Autosmart and 2 random vehicles; therefore by nature the algorithm the algorithm would lead to biased

results.

It is meant for simplicity since if not for programming the set of commands which lead to adequate behavior at intersections and on segments, such as being done for the smart vehicle; sometimes the random vehicles get into trouble, being too much off the road network.

In most cases, the simulator will self-correct their path however.

Report Approved by

Project Advisor:__Pr._Mark Campbell_____Date:_____

Project II Title:

Web Analytics: Reaching an optimal website design, search engine and ad production and customization based on linguistics, statistical information and Artificial Intelligence Methods

Author: Raphael Rubin

Abstract:

In the electronic age, identifying keywords in a document and their semantic relationships becomes a

widespread method although it existed already in the early 1940's.

To conduct a content analysis on a text, the text is coded or broken down, into manageable categories on a variety of levels--word, word sense, phrase, sentence, or theme--and then examined using one of content analysis' basic methods: conceptual analysis or relational analysis.

Traditionally, the end result is a classification of texts as books, book chapters, essays, interviews, discussions, newspaper headlines and articles, historical documents, speeches, conversations, advertising, theater, informal conversation, or really any occurrence of communicative language.

Artificial Intelligence's development has considerably benefited from Content Analysis Research.

My project is to build a search engine capable of retrieving information, as typically any other would. However, it understands the semantics, the grammatical construction, the context, and category of the website in which the relevant information is found as well as those of the information itself.

It would in an elaborate version be able to compare website contents, in real-time, extract live information, hit Google and yahoo queries, use Add Sense(the text filling utility) and based on returned adds links be able to determine what are the greatest hits and thereby improve websites and ads contents.

Report Approved by

Project Advisor:___Pr. Bart Selman_____Date:_____

Project III Title:

A platform for Statistical Arbitrage of same sector securities

Author: Raphael Rubin

Abstract:

Statistical arbitrage uses network learning technique, such as feed forward, radial basis function or recurrent NN although certain paradigms such as genetically-evolved regression models or inductive fuzzy inference systems [are also encountered in the literature.[1]

My purpose here is to reproduce the theory contained in the article by Nikos S. Thomaidis1and Nick Kondakis in An intelligent statistical arbitrage trading system and implement a system taking advantage of these price discrepancies, using neural networks and times series methodologies.

$$w_1X_1 + w_2X_2 + \cdots + w_nX_n \sim \text{mean reverting}\,(0, \sigma_t^2), \quad \sigma_t^2 < \infty$$

The different assets come in proportion w, and it is assumed that such a formula allows for a mean reverting volatility.

A standard method to identify statistical mispricing is to regress one asset against the others and test residuals. The Dickey-Fuller and Phillips-Perron unit root tests are widely used. Residuals of the regression model represent the mispricing at each time *t* of *X*1*t* relatively to *{X*2*t*,...,*Xnt}*.

We represent the dynamics of mispricing, how errors of different magnitude and size are corrected over time.

An arbitrage trading system identifies when mispricing becomes critical and positions itself henceforth *short* or *long* on an asset.

This arbitrage doesn't in principle make use of derivatives, calls, puts.

As the risk exists of a long period of time to span before the price reverts to its normal, it may be possible to

---

1

See article Nikos S. Thomaidis1and Nick Kondakis in An intelligent statistical arbitrage trading system

And the article Financial statistical modeling with a new nature-inspired technique

Nikos S. Thomaidis_1, George D. Dounias1, and Nick Kondakis1,2

cover such an arbitrage strategy with calls and puts with the proper duration and strike prices.

We will use AI methods to adjust the price of share to the index weighted average:

We will attempt in discerning and predicting the own share seasonal, timely, trend using current daily data and past weighted daily data

We will compare similar share categories and identify the market trend.

To combine Neural nets and Time series, WE take the Nikos S. Thomaidis and Nick Kondakis methods of combined *neural network*-GARCH volatility models.

Report Approved by

Project Advisor:__Pr. Bart Selman_____Date:_____

Attention: Professor Campbell

Associate Professor

Mechanical and Aerospace Engineering

Raphael Rubin

MENG ECE Cornell University

rr284@cornell.edu

DARPA Grand Challenge Project, Semester I, 2006-2007

Table of Contents

---

[2] **RICHARD W. ROTHERY, University of Texas**

I. Executive Summary:

We have achieved to create a base infrastructure for the implementation and visualization of path planning algorithms.

We have created a Neural Network class (Cbpnet) which allows to dynamically programming expected behaviors and expected types on the roads based on typical optimal scenarios.

These scenarios need to be fed at real time, while experimenting on the road.

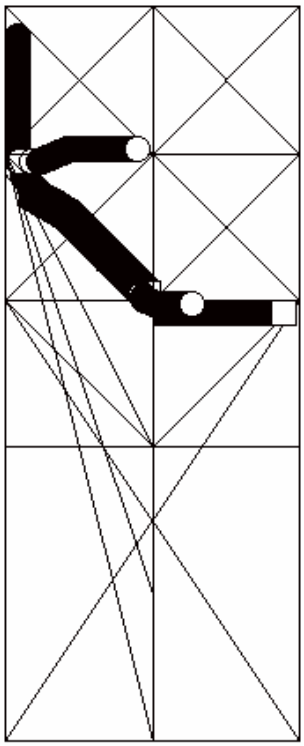
The general path planning algorithm has achieved a mixture of electronics, in the sense that at certain ticks, players run their algorithm based on input information and output certain information based on this information and priory computed information.

Also, it involves Game Theory by assigning the environment to Nature, and pre-defined states.

Players are assigned types (turn left, right..) and choose their behaviors (altruist, aggressive) accordingly. The racing game consists in optimizing road behavior based on anticipating future behavior and correct types and in a next step, by forcing players to reveal their hidden information, taking pre-emptive action to achieve such.

The goal is to have a smoothly running game, to correctly identify its competitor's hidden information and avoid accidents and penalties.

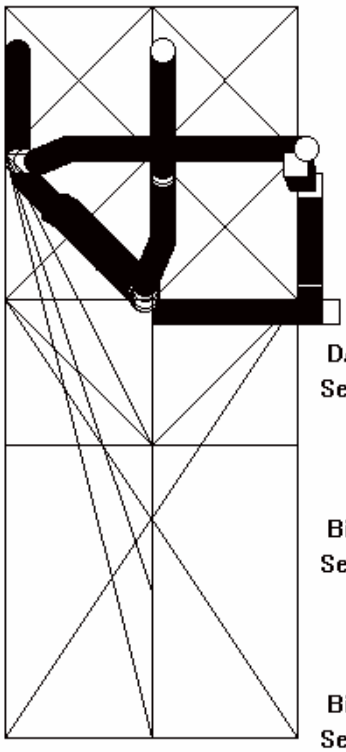
The optimal scenarios from the road experiments are to be fed into the neural network class, with 9 or more inputs and outputs.



DARPA:Position 605.200000 - 669.000000 Node: 336.000000 669.00000
Segment: 339.000000 669.000000 -> 666.000000 669.000000 Steps 267

BigWolf1:Position 397.300000 - 668.200000 Node: 336.000000 669.000
Segment: 339.000000 669.000000 -> 666.000000 669.000000 Steps 267

BigWolf2:Position 274.700000 - 337.000000 Node: 3.000000 336.00000
Segment: 6.000000 336.000000 -> 333.000000 336.000000 Steps 267



DARPA:Position 639.153120 - 332.334768 Node: 669.000000 336.00000
Segment: 670.101958 333.209715 -> 334.898042 5.790285 Steps 30286

BigWolf1:Position 334.100000 - 94.800000 Node: 336.000000 336.0000
Segment: 336.000000 333.000000 -> 336.000000 6.000000 Steps 30621

BigWolf2:Position 664.000000 - 337.000000 Node: 336.000000 336.000
Segment: 339.000000 336.000000 -> 666.000000 336.000000 Steps 306

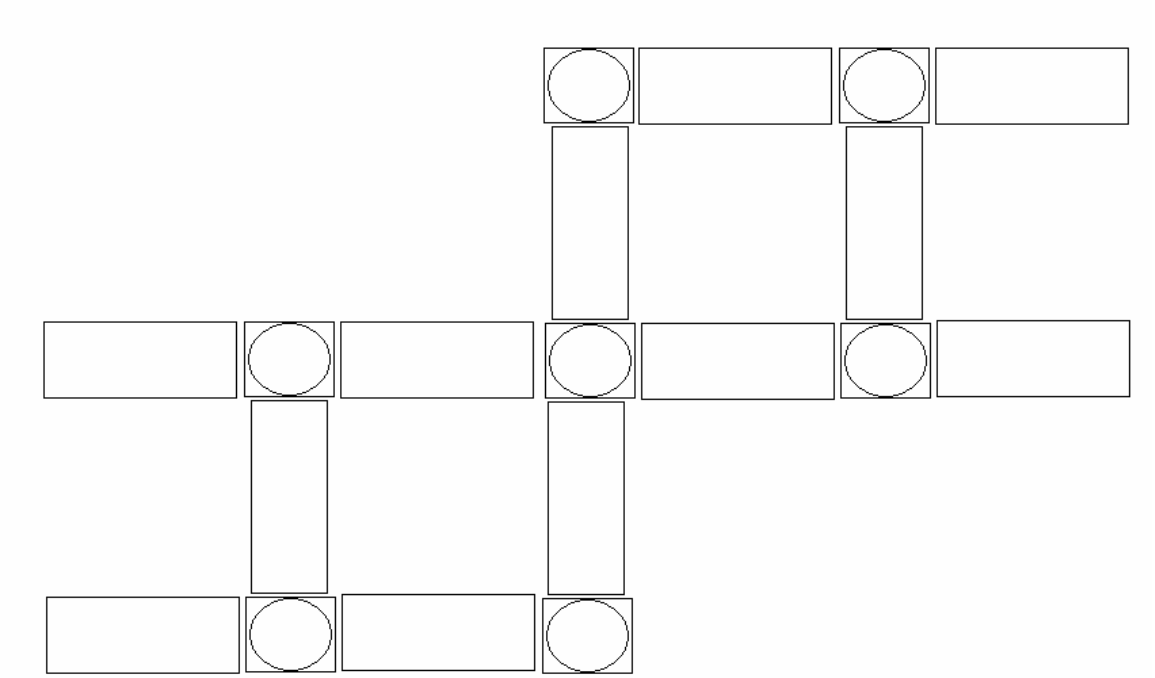
II. **Simulator Components**

In theory, the simulator reads a RNDF file and figures the absolute position in the world of every intersection (node), creating henceforth connected segments with beginning and end points and all specifications such as lanes number, spacing, and stop signs.

In practice, we have thought it was not necessary in a first approach to actually use a real data network file since we do not have yet an accurate description of vehicle dynamics, and sensors data, and therefore may not match expected curvature to the observed one.

Yet, we have created different possible shapes for the segments and therefore end points could be easily matched to a function returning end points and a set of sampled points along the cubic spline curvature.

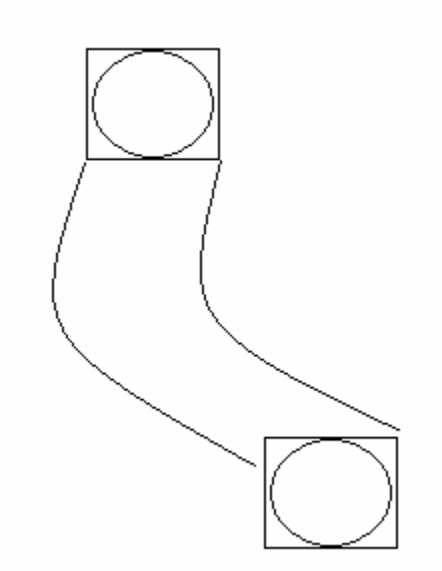
Initially the world is created as straight segments and square shaped intersections whose beginning and end points interconnect.



Road Network 1

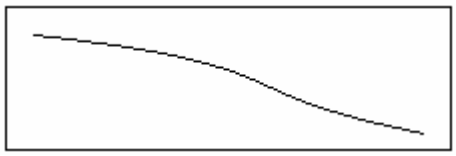
Therefore, such a design doesn't change in any way the path planning methodology.

In order to adapt this technique to more sophisticated curves, in the case of splines, we still have beginning and end points, boundary boxes for lane spacing, orientation of the vehicle.



Spline Segment

The only difference with a spline segment is that the calculation of the boundary boxes is calling the spline function to make sure the vehicle is still in its rightful position (center of the lane, in the case it is in the state idle), and to identify the precise path of the trajectory to change lane (function returning a path_extrapolation vector of where WhereIam class, containing attributes such as the future positions to be at, and the time to be at them, while the change of lane or change of intersection is being executed.

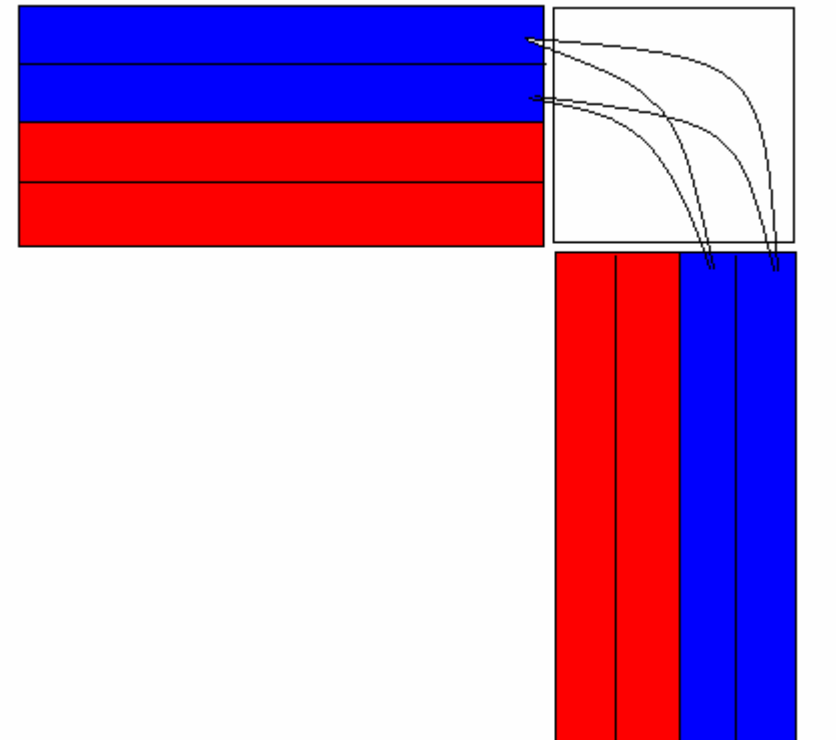


Regular Lane change 1

Indeed, we mostly worry only about x and y components of speed and position and bounding boxes calculations are easily performed.

One of the issues is finding the orientation of the vehicle, left to right, or top to bottom, in order to identify its heading.

Indeed, when the vehicle operates a change of lane, or a turn at an intersection, we add an x, or a y component to the speed depending on the heading, until the circular arc, or the arc is completed and the vehicle finds itself in the new lane, or the new segment.



Turning from lane1 to 1, 1 to 2,2-1,2-2

The components of an Advanced Simulator would be the True Road Model, the Sensor Model, the Pose Tracker, Vehicle Dynamics, Actuator Commands, as well as the Behavioral Modeling of Traffic Vehicle Simulation and Collision Detection.

In the simpler version of the Simulator, we have implemented only the True Road Model, Simplified Vehicle Dynamics, Actuator Commands, as well as the Behavioral Modeling of Traffic Vehicle Simulation are designed and implemented.

The Geometry Specification consists of line segments, circle segments, splines, and clothoids to determine curvature. The simulator should not have to worry about computing curvatures; the driver just knows the segment attributes, it is on, and executes commands checking the necessary bounding boxes, using the latter attributes.

Instead of going straight as we implemented it here (setting all the time either speed_x or speed_y to 0), the driver would follow the path extrapolated vector of points indicating to it what its next moves should be.

Therefore it computes its x and y instantaneous velocities using the next key point position, its current position.

A cubic spline is defined as a polynomial.

$$S_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i \quad \text{for } x \in [x_i, x_{i+1}].$$

Therefore the vehicle, which travels at a certain speed_x, and speed_y computes the next extrapolated points it should reach at a certain time stamp defined in constant.h

If it receives a command to change speed, it modifies its next extrapolated points so that each new extrapolated point (whereIam Class) corresponds to each time stamp, until the end of the segment, and the reaching of the next intersection.

The Vehicle Dynamics take actuator commands (steering, brake, and throttle) and determine angular rotation rates and accelerations (simple acceleration models).

We purposefully chose to ignore vehicle dynamics just using specific time stamps to update the vehicle's position using the commands generated p_event data (please see annex).

The true world model instantiates the graph, vehicles, drivers, obstacles, lanes, nodes (intersections).

The sensor model consists of attributes on each class elements which are to be compared to the expected attributes values returning an eventual error in the feedback to the upper Artificial Intelligence layers: indeed as the simulator processes the commands and the sensors data, it returns a feedback which will be fed to the upper layers of path planning.

Collision detection is performed through bounding boxes.

A Bounding box returns true if a vehicle is in a specific lane, or at a specific intersection.

Therefore the AND operation of two bounding boxes will assure that no two vehicles will collide, for example, and this computation allo9ws to tag visible neighboring cars and obstacles.

The Paths for changing lanes and turning left and right are set such that the vehicle performs a smooth trajectory, or a perfectly arc shaped one in the case of an intersection. The driver.cpp file (see annex) is responsible for most of the handling of the control systems, finding vehicle orientation, direction, position, neighboring obstacles and vehicles (to be tagged).

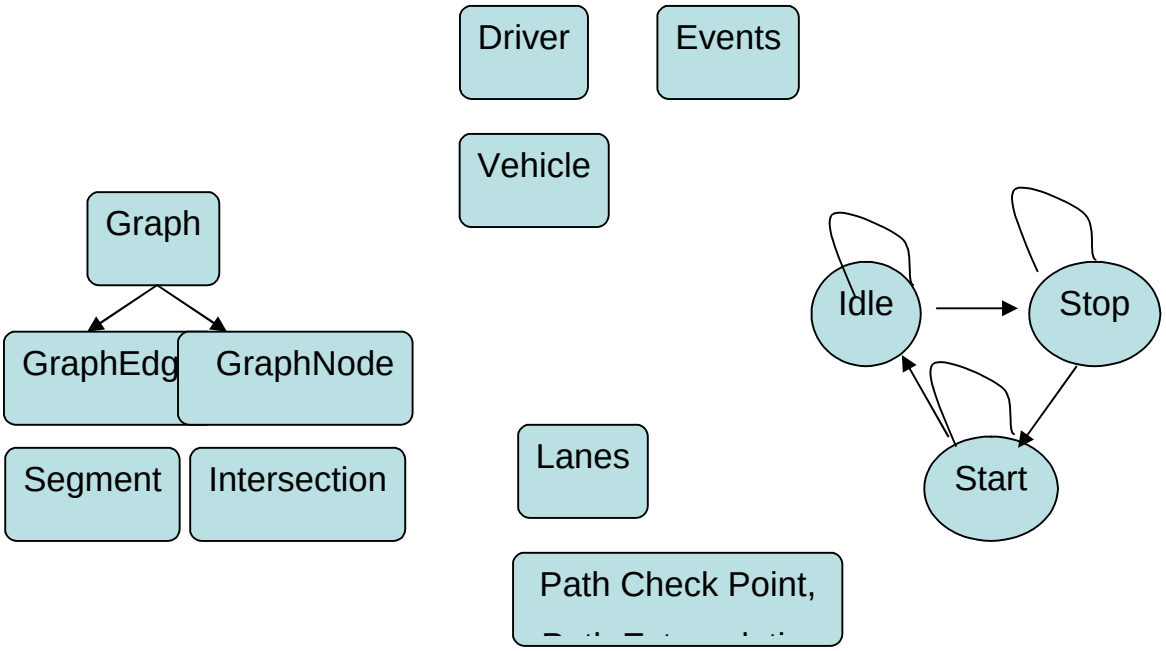
The behavioral model consists of one DARPA vehicle abiding to a set of commands generating external events and internal events (see annex) as well as other vehicles running randomly in the graph.

No prediction in performed at this level except for checking that at an intersection you may not press forward without a renewed instruction, or that you are not transgressing a safe following distance, speed limitations, or penetrating the bounding box of a vehicle.

III. Simulator Design and Implementation

[3]

The implementation is exposed in the Annex; here we illustrate the basic class hierarchy.



The graph is a connected graph and from its basic objects such as nodes and segments, it derives lanes and intersections.

A driver implements the state machine class since the driver expects external events (commands).

These commands can only be processed in a certain state. For instance it is not possible to change lane in an intersection.

One needs to be idle to change speed; therefore after a Stop command, you reach the Stop state. The only way to change speed in to first goes back to the start state.

The states will be permanently executed until the end of the state function, and then switch back to idle state; updating its position using default parameters.

---

[3] The implementation is thoroughly covered in the annex.

The vehicle to which the drivers points to, points itself to the driver, its position in the world, its neighbors…

The State machine is made of turn left, turn right, change left, change right, stop, start, idle states. While the internal functions associated to the state execute, a check is performed that it is the only function required in this state.

In GraphTest.cpp we have created all the world's elements and allowed for the driver to process a command.

This code checks if driver has reached an intersection and issues a new command since the previous one is systematically destroyed.

```
while(DARPA->intersection!= true){
DriverData* pData = new DriverData;
    pData->speedx=0;
pData->speedy=1;
pData->lane_index = 2;
toto->SetSpeed(pData);}
```

In consequence, once the intersection has been reached, the vehicle remains in the idle state, with speed 0.

If we wish to turn left once the intersection has been reached, we would just call on toto->TurnLeft(pData);

IV. Behavioral Specifications

The Behavioral model of traffic fixes previous vehicles positions and actions detected.

It is meant to avoid collisions, to reach safely the next intersection and check point, in the shortest time.

Therefore it has to constantly at one critical moment take the decision of infringing or not the road regulation, at the expense of a penalty inflicted by the DARPA organizers, but with the benefit of saving time and gaining

predictability.

The Sensor Model of the true world must be reliable.

Drivers Behaviors are Erratic, Cut-off, Route follower, yet we must precisely quantize their behavior.

We are using artificial intelligence Nash equilibrium in incomplete information, coupled with the theory of artificial intelligence, neural networks.

The players decide whether the players are aggressive (ready to infringe road regulations to get there faster) or altruist (respect of the other, and right of way).

Thus every player knows its behavior but ignores that of others, it has to compute expectations to have a better idea of the player's behavior.

Nature decides what the type of the players is: turn left, turn right at the next intersection.

Therefore the behavioral model splits the game in 3 steps:

Nature plays and decides on the type; then the player 1 plays and decides on its behavior, having yet no information on the other players, since one clock cycle has not passed.

At the next clock cycle, nature decide for players 2 and player 2 decides for its type even though the players 2's decision will be influenced by the contradictions observed between the expected and observed behavior of player 1 and its expected type (not observed).

Vice Versa, the player one has now some information and updates its weights in the neural network regarding its expectations on the other players' types and behaviors.

Therefore at every clock cycle, the expectations will be reinforced and once the probabilities are reaching a sufficient value, the action of bypassing the other vehicle on the left, or patiently wait in the right lane is decided.

Being behind a vehicle before turning in an intersection may be dangerous since the visibility at the intersection is reduced and therefore passing it may be a wise choice if the vehicle is being aggressive or conveys too many contradictions between its expected types and behaviors.

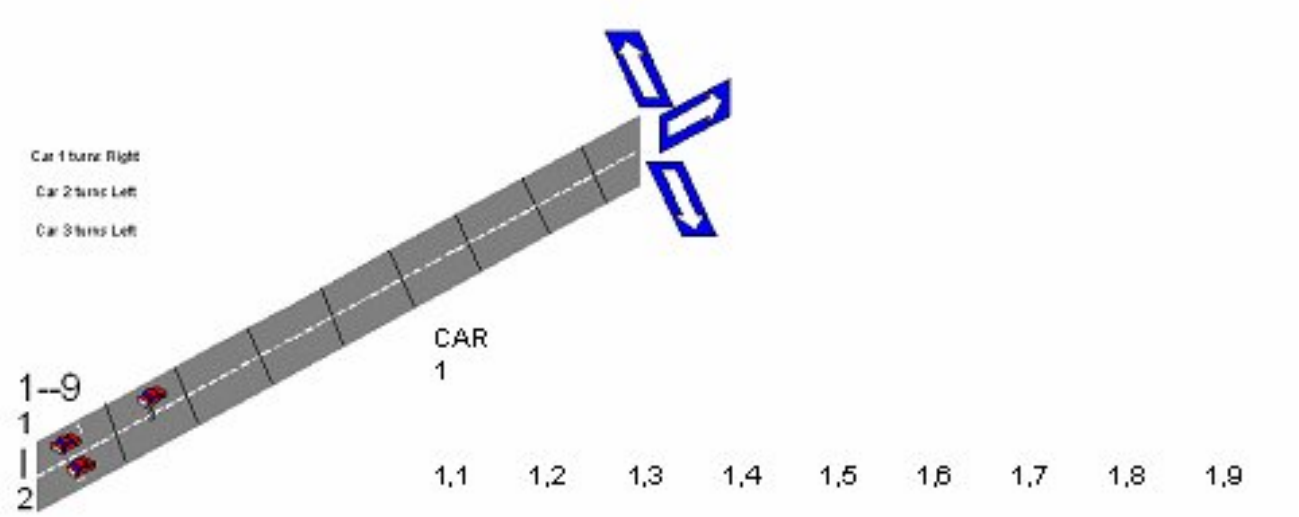
The gain will change only if at some point, the player is impeded to go in the right lane before the intersection

or at the intersection.

Whereas, he could have avoided this situation by moving into the right lane sooner while violating the traffic regulation (penalty).

A traffic violation is a further indication of a certain behavior as well as destination.

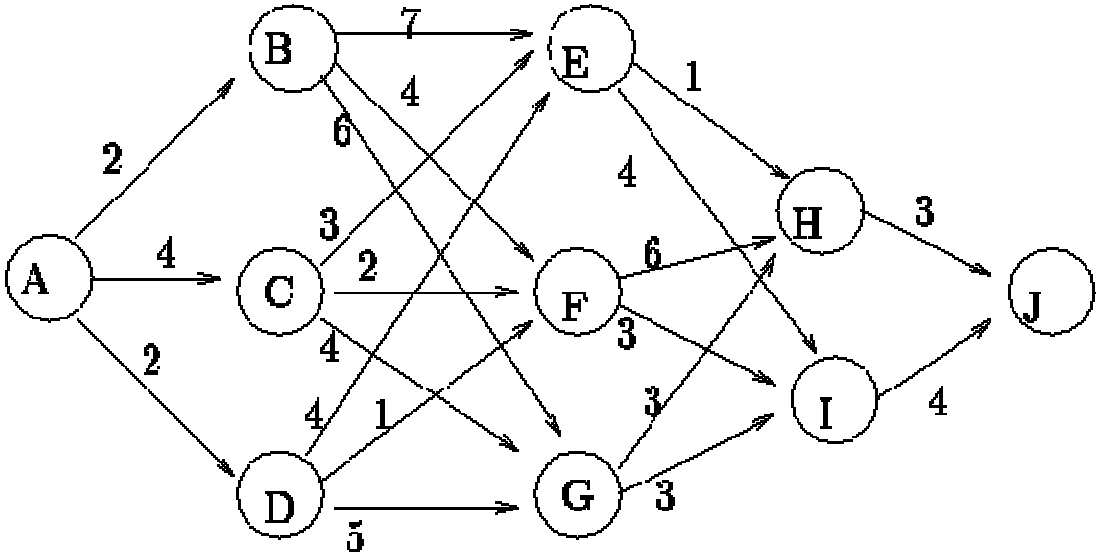
Therefore, the whole point is to accurately identify the expected destination of the other cars on the roads as well as their 'type' of driving.As a result, the looping on the tree will produce better and better expectations.



The prisonner's Dilemma 1

As we shall see very formally soon, the problem is very much like a Prisoner’s Dilemma where the players are incited or not to cooperate, according to classical Economic Theory.

The dynamic programming approach may not be relevant here since only one action has to be taken here: decide at some point to pass the vehicle on the left or just give it the right of way.



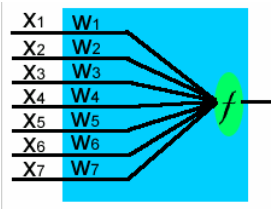
Dynamic Programming 1

$$g(w) = \max_{j} (b_j + g(w - w_j))$$

Such an approach somewhat similar to taking the shortest path processing for the destination node (intersection) to the source is possible but it ignores the behavioral aspects of the vehicles and the necessary contradictions observed between expected types and behaviors, so critical for path planning prediction.

V. Behavioral Design

We Use a Perceptron Model at Each Gate



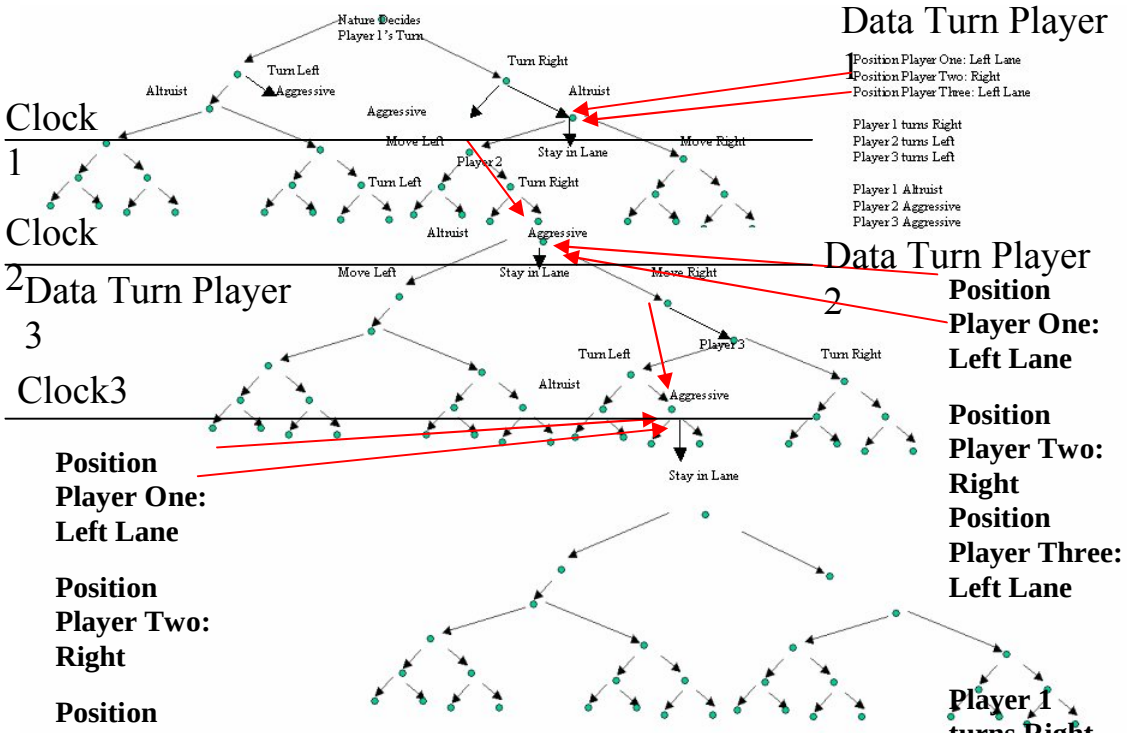
At the output, below a certain threshold

- $w(j + 1) = w(j) + \alpha(\delta - y)x(j)$
- δ =expected output,
- α is a constant and $0 < \alpha < 1$ or learning rate
- y denotes the output
- $w(j)$ denotes the j-th item in the weight vector
- $x(j)$ denotes the j-th item in the input vector

[2] We are using a 3-clock cycle for every turn. That is when 3 players will have completed their first turn, 3 clock cycles will have completed.
The clock in itself is figurative. It stands for a separation and a chronology in the processing of the data.

Let us actually visualize the process:

[1] Wikipedia: Perceptrons



Position Player One: Left Lane

Position Player Two: Right

Position Player Three: Left Lane

Player 1 turns Right

Player 2 turns Left

Player 3 turns Left

Position Player One: Left Lane

Position Player Two: Right

Position Player Three: Left Lane

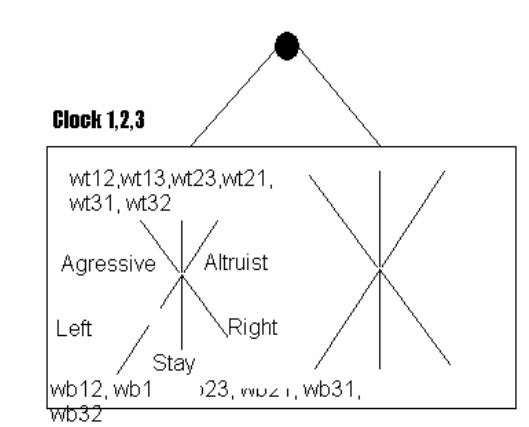
Player 1 turns Right

Player 2 turns Left

Player 3 turns Left

Basically, at every turn the entry of a gate is the sum of its weighted inputs. The red arrows derive from previous cycles; in case of a contradiction between a type and a behavior, weights are being changed.

Let us examine this graph more carefully:



We have replaced every decision node by a sum of specifically weighted inputs.Let us formalize the set of problems the vehicles face, in traffic, in such a way that we may compute the necessary weights.

Indeed, we are going to train our network so as to minimize the errors beteween every player one 's expectation of other player two 's behavior wb12…(Turn Left at next Intersection, Turn Right at next intersection, stay in lane) and every player one 's expectation of other player two 's type wt12…(Aggressive or Altruistic).

There is knowledge only known by the individual driver, RealBehavior1 = Right at next intersection, RealBehavior2 = Left at next intersection, RealBehavior3 = Stay in Lane at next intersection.

There are Type Clauses, General Behavior Clauses, and Reasonning Clauses.

Such clauses are necessary to compute the action taken at every step, to compute one's actual type, to determine expectations about other players behaviors or types.

Indeed, les us see samples of some pseudocode:

```
left_lane = 0;

right_lane = 1;


%axis center is bottom left corner (let lane)

position_player_1 = [0 0];

position_player_2 = [1 0];

position_player_3 = [1,1];


%go left 1, stay in lane 2, go right 3

GoLeft = 1;

StayInLane = 2;
```

GoRight = 3;

Altruist = 0;

Agressive = 1;

RealType1 = altruist;

RealBehavior1 = StayInLane;

RealType2 = altruist;

RealBehavior2 = StayInLane;

RealType3 = altruist;

RealBehavior3 = StayInLane;

Action_taken_1 = GoLeft;

Action_taken_2 = GoRight;

Action_taken_3 = StayInLane;

behavior_anticipated12 = StayInLane;

behavior_anticipated13 = StayInLane;

behavior_anticipated21 =StayInLane;

behavior_anticipated23 =StayInLane;

behavior_anticipated31 =StayInLane;

behavior_anticipated32 =StayInLane;

type_anticipated12 = altruist;

type_anticipated13= altruist;

type_anticipated21= altruist;

type_anticipated23= altruist;

type_anticipated31= altruist;

type_anticipated32= altruist;

%Agressive means that it goes over the speed limit.

%No notion of Utility or Cost is introduced, as we will see later.

This set of weights if the expected behavior of player 1 of player 2 etc… and the expected type of player 1 of player 2…

initial = 0; wb12 = 0.5; wb13 = 0.5; wb21 = 0.5;wb23 = 0.5; wb31 = 0.5;wb32 = 0.5;
wt12 = 0.5;wt13 = 0.5;wt21 = 0.5;wt23 = 0.5; wt31 = 0.5;wt32 = 0.5;

The update is being made here after the algorithm has run, for each clock tick;

wt12 = wt12 +training_rate*(RealType2 - type_anticipated12);

wb12= wb12+ training_rate*(RealBehavior2 - behavior_anticipated12);

wt13 = wt13 + training_rate*(RealType3 - type_anticipated13);

wb13= wb13 + training_rate*(RealBehavior3 - behavior_anticipated13);

Once the errors on the weights have been computed, we proceed to the actions taken and type chosen by each player, using the formalism below, amounting to a simple and operation of all the clauses.

Finally, the expected behaviors and types are computed

%Next we update the player's position

position_player_1 = UpdatePosition(1,Action_taken_1, position_player_1);

Part I of the Formalism:

Type Clauses:

1) If x can not go left and ActionTaken by x = Left,
if y is altruistic at time t and altruistic in t-1 (and y is the blocking vehicle), x becomes aggressive in t+1.

2) x is aggressive if y is expected to be aggressive and y is altruist.

3) x is aggressive if y is expected to be altruistic and y's behavior is altruisitc.

4) x is aggressive (more and more) if conflict between x's RealBehavior and Position during the time left to the Intersection (Decision).

It means that the older conflicts are weighed less heavily than the newer ones

This is where we use our conflicts class..

5) x is aggressive is y's t-1 move contradicts y's t-2 move.

6) x is altruistic if y is expected to be altruistic and y's expected behavior is different of x's.

7) x is altruistic if y's t-1 move doesn't contradict y's t-2 move.

Part II of the Formalism:

General Behavior Clauses:

1) If x is altruistic => x goes (Left, Right, Stay in Lane) if RealBehavior of x = (Left, Right, Stay in Lane) and if x can go (Left, Right, Stay in Lane) and if y is expected to be altruistic.

2) If x is altruistic => if y is expected to be altruistic and y's expected behavior conflicts with RealBehavior of x, then x stays altruistic.

Transition Clause:

3) If x is altruistic, if conflict in t-1 and t-2 between the players, x becomes aggressive.

Safekeeping Clause:

4) If y is aggressive be altruistic.

We do no want accidents.

Part III of the Formalism:

Reasonning Clauses:

1) x expects y to be aggressive if y is aggressive in t-1 (ponderation by e-ro*t-1), t-2 (ponderation by e-ro*t-1)…

if y's expected behavior conflicts with x's Real Behavior.,

if y's expected behavior conflicts with y's actions.

2) x expects y to be turning (Left, Right, Stay in Lane) if y's position in on the (Left, Right, Stay in Lane).

References:

Artificial Intelligence class CS 472, Cornell University.

Game Theory class, University Paris I Pantheon-Sorbonne.

DARPA Urban Challenge Class 1.0 Reference Manual

Raphael Rubin, 2007, USA

```
#define TIME_SLICE 1 //1 second time slice
#define INITIAL_TIME 1
#define SPAN 10 //Defined for Bounding measurements
#define RAD_DEG 2*3.14/360 // Defined for conversion between rad and degrees
```


External Methods, called by Driver():

```
void Halt();
void SetSpeed(BaseGameEntityData*);
void SetAcceleration(BaseGameEntityData*);
void SetAngle(BaseGameEntityData*);  //relative to map coordinate axis
```

State machine state functions

```
    void ST_Idle();
    void ST_Stop();
    void ST_Start(BaseGameEntity*);
    void ST_ChangeSpeed(BaseGameEntity*);
    void ST_ChangeAngle(BaseGameEntity*);
```

Members:

Each entity has a unique ID

```
  int         m_ID;
```

Every entity has a type associated with it

```
  int         m_EntityType; //Moving, Static...
```

Used by the constructor to give each entity a unique ID

```
  int NextValidID(){static int NextID = 0; return NextID++;}
```

Its location in the environment

```
Point m_vPos;

Shape shape;
Speed_x;
Speed_y;
Acceleration_x;
Acceleration_y;
double Angle;
```

State map to define state function order

```
BEGIN_STATE_MAP

END_STATE_MAP

    enum E_States {   ST_IDLE = 0,ST_STOP,ST_START, ST_CHANGE_SPEED_AND_ANGLE,
ST_CHANGE_SPEED, ST_CHANGE_ANGLE, ST_MAX_STATES};
```


Public Members:

Private Members:

```
Weights for the neurons INPUTS.   STATIC because shared
static  float input_Weights[9][9];
float behavior_Weights[2];
Weights for the DATA logic for each player.
float type_Weights[2];
Weights for the DATA logic for each player.
float Sigmoid(float);
The sigmoid function.
```

Methods:

```
These two functions train or run the net. Both return
the result that the net gives. The final variable that
Train() takes is the expected value.
float Train(int, int,int, int, int, int, int, int, int,int,int,int);
  Inputs =>
  offset 0, offset 1, offset 2 lane 0 , lane 1, lane 2
  my known type, other expected type1, other expected type2, my known behavior,
  other expected behavior1, other    expected behavior2


  Updates: if not specified, types and behaviors and actions are similar
  theApp.toto->autosmart->information.behavior_1[offset0] = theApp.toto-
  >autosmart->information.behavior_1[offset0 -1];
  A driver points to its information class. (see Information Class below)
       Definition and detection of agressiveness
       WEam not able to change lane
       WEam being chased, or slowed down
       int index_block_front =0;
       int index_block_left =0;
       int index_block_right =0;
       intindex_responsible,offset_victim,offset_culprit,conflict_type

We instanciate all these variables, the index of the vehicles that are blocking
the vehicle from accelerating in its lane, or turning left or turning right, by
calling the appropriate functions.

We then run the Type Clauses, Behavior Clauses and Resaonning Clauses.

Example:
If x can not go left and ActionTaken by x = Left,
if y is altruistic at time t and altruistic in t-1 (and y is the blocking
vehicle), x becomes aggressive in t+1.


  if (index_block_left != 0 && theApp.toto->autosmart-
>information.action_1.back()== left_lane){
/*1*/    if(theApp.world->m_Vehicles[index_block_left]-
>information.behavior_1[offset0] == altruist && theApp.world-
>m_Vehicles[index_block_left]->information.behavior_1[offset0-SPAN] == altruist)
       theApp.toto->autosmart->information.behavior_1[offset0 + 1]= agressive;




float Run(float, float);
```

Methods:

BOOL CDARPAApp::InitInstance()

```
GraphNode *A;
```

Beginning, end points and the Circle's center if the shape of the intersection is a center.

We then connect the nodes where only needed is the radius(3) and the absolute angle(90).

```
C2->connectTo(C3,   Straight(3, Point(0,0), Point(0,0)  ) , 90.);
```

We instanciate drivers, vehicles, push back these entities into the world, set up their lane index.

Members:

Autosmart is the name of the vehicle to which every driver(3) points to. Although the individual drivers names are toto(smart driver), titWeand tata(automatically programmed) and the individual vehicle names are (BigWolf1, BigWolf2 and Autosmart)

Methods:

It draws segments throughout the graph edges.

```
void CDARPAView::drawGraph(CDC* pDC);
```

```
class MyParameters; whose purpose is to pass along with the   afxmessagethreads
the required drawing context parameters and commands.

void check_for_beginning(void);
It is a fix that allows to reset position at the initial node and throws the
vehicle into the intial segment from node A to node B.
```

```
void drawRandom_call(MyParameters * parameters){};

This function is being called up in a while loop throughout the duration of the
"racing game" and ONLY by the non smart drivers.

Random integers determine whether the vehicle should turn right or left. Then
vectors store all possible headings if the vehicle should turn left and viece-
versa for the right.
```

Then, if the heading is made up of 1's, a further distinction must be made beccause several segments in between quartets angles may bear that type of headings. Another random integer makes the distinction.

```
vector<int>heading_x_left;
vector<int>heading_y_left;
vector<int>heading_x_right;
vector<int>heading_y_right;
  IsPossibleTurnLeft
  IsPossibleTurnRight


  goto turn_left;
  goto turn_right;
```

This is an example of a for loop iterator over possible headings.

```
     for (vector<int>::const_iterator it(heading_y_right.begin()); it !=
     heading_y_right.end(); ++it) {
     temp+=1;
     headings_right_y = (*it);
     if(temp == random_right)
     break;
     }
     temp =0;
```

This is an example of a for loop iterator over more sophisticated headings_left, in case several segments are of 1's type of headings.

```
else if (ispossibleright == false && ispossibleleft == true){
  for (vector<int>::const_iterator it(heading_x_left.begin()); it !=
heading_x_left.end(); ++it) {
    temp+=1;
    headings_left_x = (*it);
  if(temp == random_left)
     break;

        goto turn_right;
```

This code will then browse through the obtained random and AVAILABLE headings, retreive the future expected segment, pass the proper comand, and once out of the intersection penetrate the segment till its extremity.

```
if(headings_left_x == 1 && headings_left_y == 1 ){
pData->speedx = 1;
pData->speedy=  1;
pData->lane_index = 2;
theApp.titi->ExternalEvent(2, pData);
theApp.titi->autosmart->heading.x = headings_left_x;
theApp.titi->autosmart->heading.y = headings_left_y;
SegmentType * segment = theApp.titi->autosmart-
>node.turnRight(headings_left_x,headings_left_y,theApp.titi->autosmart,
index_of_1_1);
if (segment == NULL )
goto start;

theApp.titi->TurnLeft(pData,index_of_1_1);
while(theApp.titi->autosmart->intersection == false)
{
theApp.titi->ExternalEvent(2, pData);
```

```
Sleep(10);
}
```

```
UINT  drawToto(LPVOID param){}
```

This is the programmed version of the code aboce; it is more reliable as careful commands are being passed at the intersections in accordance with positionning; the vehicle (autosmart) piloted by Toto, never gets in trouble, as a result of its commands.

Autosmart is performing a continuing closed loop.

```
void CDARPAView::OnDraw(CDC* pDC)
```

```
     CWinThread *pthread1;
     pDC = this->GetDC( );
```
This returns the drawing class context

```
     //pDC = pDC_temp;
     //this->pDC = pDC_temp;
     //CwinThread *pThread = AfxBeginThread( ThreadFunction, &data);
     LPVOID data =0;
     int test = 5;
```
AfxBeginThread(drawToto,pDC);
This function calls the above mentionned function along its context parameter.
```
     Sleep(1000);
     AfxBeginThread(drawRandom,new MyParameters(0, 1, 1, 1, "turnleft",this));
     Sleep(3000);
     AfxBeginThread(drawRandom1,new MyParameters(0, 1, 1, 1, "turnleft",this));
```

```
       while(1){
       drawGraph(pDC);
       pDC->FloodFill(30,30,RGB(12, 3, 125) );
       pDC->Rectangle(MAR_X+theApp.toto->autosmart-
       >position.x/4,MAR_Y+theApp.toto->autosmart-
       >position.y/4,MAR_X+15+theApp.toto->autosmart-
       >position.x/4,MAR_Y+15+theApp.toto->autosmart->position.y/4);
       pDC->Ellipse(MAR_X+theApp.titi->autosmart-
       >position.x/4,MAR_Y1+theApp.titi->autosmart-
       >position.y/4,MAR_X+15+theApp.titi->autosmart-
       >position.x/4,MAR_Y1+15+theApp.titi->autosmart->position.y/4);
       pDC->Ellipse(MAR_X+theApp.tata->autosmart-
       >position.x/4,MAR_Y2+theApp.tata->autosmart-
       >position.y/4,MAR_X+15+theApp.tata->autosmart-
       >position.x/4,MAR_Y2+15+theApp.tata->autosmart->position.y/4);
```

This part displays the vehicles shapes on the graph, rectangles, ellipses.

Finally the textual messages:

```
       sprintf(toto," DARPA:Position %f - %f Node: %f %f", theApp.toto-
       >autosmart->position.x,theApp.toto->autosmart->position.y,theApp.toto-
       >autosmart->node.getPosx(),theApp.toto->autosmart->node.getPosy());
```

Driver inherits from the state machine class. It is instantiated by its ID, the path extrapolation and the path checkpoint. It points to the world, external events, bounding functions, the state machine functions.

A typical scenario consists of an external event being generated, which, again, boils down to a function call into the class's public interface. Based upon the event being generated and the state machine's current state, a lookup is performed to determine if a transition is required.

If so, the state machine transitions to the new state and the code for that state executes. At the end of the state function, a check is performed to determine whether an internal event was generated. If so, another transition is performed and the new state gets a chance to execute.

This process continues until the state machine is no longer generating internal events; at which time the original external event function call returns. The external event and all internal events, if any, execute within the caller's thread of control.

Once the external event starts the state machine executing, it cannot be interrupted by another external event until the external event and all internal events have completed execution.

This provides a multithread-safe environment for the state transitions. Semaphores or mutexes can be used in the state machine engine to block other threads that might be trying to be simultaneously access the same object.

The vehicle’s heading will be in the set: {1,0 -1,0 0,1 0,-1 } and {1,1 -1,1 -1,-1 1,-1 } corresponding to horizontal or vertical directions, where we consider that a set of one's in taken in absolute value signifies that the absolute angle to the standard Cartesian coordinates in neither 0. nor. 80, nor 180. nor 270..

The Driver’s constructor consists of the driver’s ID, the Path Extrapolation (class vector of points to go and at what time to in the case a function returns the spline points at some stamp values), and the Path Checkpoint (class vector of all points to reach and at what time, during the challenge).

Following external events (commands) such as Start, Halt, SetSpeed, SetAngle, SetLaneLeft, SetLaneRight, TurnLeft, TurnRight, the transition map tables of states is processed, looking for the necessary transitions.

The Bounding Box 1 checks that the vehicle is indeed in the segment that has been passed on the argument. (The computation depends on the heading). On an oriented arc the vehicle may be heading towards the beginning point of the edge or towards the edge point. When the edge is not a straight segments but a spline, this computation still uses normalized headings (the tan of the heading vector will indicate the quadrant {1,0 -1,0 0,1 0,-1…} the vehicle ‘s heading corresponds to.

The heading is necessary since the beginning and end positions vary depending on the direction and sense of the vehicle.

The control statement if (posX <= edge->beginning->getPosx() && posX > edge->end->getPosx()) would become if posX or posY belongs to the center point Edge spline +/- lane spacing on both sides.

The Spline class has not been implemented, however Windows does provide readymade classes for spline extrapolation when given beginning and end points and a number of points over which to draw segments. Thus, a possibility is the cutting-off of the spline-shape road network into a multitude of segments which our simulator knows well to handle.

Bounding Box 2 checks that the vehicle is at a given intersection.

Update() is an essential; function since it will be called by all Vehicle's state machine methods to determine when the transition occurs between a segments and an intersection and vice-versa.

It is the component which introduces intelligence into the simulator by obliging the driver (us) to provide the proper driving instructions on the segments and a intersections; otherwise it gets lost or just stays stuck at an intersection because it doesn't find the next segment.

Update() determines first in which lane, which segment, which intersection the vehicle is in: given the position of the vehicle, it goes through all edges and nodes and checks bounding box 1 and 2; the possibility of accessing the world’s elements such as the graph structure, derives from the fact that the Driver points to a Vehicle and to the World. The result of this loop is to set the intersection or the segment the vehicle is at.

As mentioned earlier, Path_extrapolation contains a whereIam class (points and times of passage); Update sets its present and future values of positions and velocities, or even segments and lanes.

It computes the offset; For a spline, one would calculate the distance to the beginning point or end point using an arc length function.

The heading is recalculated at each passage of Update(), just like the current segment, the current intersection, the current heading.

Update updates positions using simple vehicle dynamics.

An offset on road segments allows to compensate for improper positionning by readjusting to the center of the lane.

The external events aforementioned give rise to the processing or new states, or to the same state with different data.

ST_Idle maintains the speed recorded in the previous command p_data and returns the Update function with unchanged values;

ST_Stop and ST_Start won't run the vehicle but are transit states and stopping states.

The vehicle must start before being idle.

ST__ChangeAngle changes the angle.(if needed)

ChangeLaneLeft, ChangeLaneRight, TurnLeft, TurnRight are the most complicated functions of the file driver.cpp. First we determine the lane the vehicle is on and the lane the vehicle is heading towards; we check for the markings, depending on the heading of the vehicle, we add a x or a y component to the velocity to accomplish the turn, until the current position is that of the future point we must be onto, or that we have reached the segment we or the lane desired.

(calling the Update() function accordingly)

Turn Left fist determines whether we are at an intersection, then returns the segment which corresponds to the current headings; pData may include the actual index of the segment we which to turn on, as the call to the function isPossibleTurnLeft() in Specifications.cpp would encompass all the segments that are available on one's left.

We analyze the road from a driver's standpoint and not a machine's stand point; indeed the path planning algorithm uses human reasoning and would not be efficient otherwise.

Eventdata.h

This is the command data to be passed on to an event.

It allows to Add Graph Nodes, Add Graph Edges once the node has been added.

It allows to connect a node to another specifying its shape.

```
class Conflict
```

public members:

```
int index_responsible;
int offset_victim;
int offset_culprit;
int conflict_type;   //turnright,turnleft,stay
```

Keeping records of conflicst is an essential part of the path planning algorithm as new conflicting situations idnetified may be linked to older conflicts, or to determine patterns.

```
enum lanePosition{leftLane,RightLane};
enum types{turnright,turnleft,stay};
enum behavior{altruist,agressive};
enum players {toto,titi,tata};
enum action_taken{left_lane, stay_lane, right_lane};
```

Enumerations allow to simplify the algorithm by allowing to work with strings.

Every Vehicle has information on every one else's positions, provided they are located on the same segment.

It could also have expected positions, for example in the case a blocking vehicle has been identified but is clearly hidden, currently by another vehicle; Working with expected positions would avoid loopholes.

Every Vehicle has information on every one else's lane positions, provided they are located on the same segment. It knows its type. It observes everyone's behavior and actions taken.

It computes anticipated types and behaviors.

```
class Lane
```

Members:

```
Each valid lane of a segment has an index
int m_iIndex;
double spacing;
bool direction;
Use enums for types of markings :
enum markings{no_markings, continuous, dotted_marking};
bool markings_left; //used in ispossiblechangeLeftlane
bool markings_right; //used in ispossiblechangeRightlane
VehicleVector VehiclesInLane; //used in ispossiblechangeleftlane
by providing the number of vehicles on thelane
```

```
class SegmentType : public GraphEdge{}
```

```
Members:
```

```
A segment is made up of shape, angle, headings, lanes, offset to beginning,
vehicles on segment.
```

Methods:

```
SegmentType * turnLeft(double , double, Vehicle * );
Returns the GraphEdge which is the most on the left
SegmentType * turnRight(double , double,Vehicle *,int);
Returns the GraphEdge which is the most on the right
bool IsPossibleTurnLeft(double , double ,Vehicle *);
bool IsPossibleTurnRight(double , double,Vehicle *);

int IsPossibleTurnLeft(double , double ,Vehicle *, int);
int IsPossibleTurnRight(double , double,Vehicle *,int);
```

Members:

```
          Point position;
          Vector heading ;//normalized to vector unity
          Vector perpendicular_heading;
           static Vector map_heading_vertical ;
           static Vector map_heading_horizontal ;
          double    angle;//to the vertical heading
         double angular_rotation;
         double velocity_X;
          double velocity_Y;
          bool random;
          int prev_segment_headx;
         int prev_segment_heady;
         int prev_segment_angle;
         const string name;

         int steps;
        // double    mass,
       //  double    max_force,
          static double    max_speed ;
          static double    max_turn_rate;  //90 degrees per second;   using 50ms
timestamp
         Circle circle;
           Lane lane;
           SegmentType segment;
           Intersection node;
           bool intersection;
           int lane_index;
```



```
Point c1,c2;   //world dimensions

A container of all the moving entities
  std::vector<Vehicle*> m_Vehicles;

Any obstacles
std::vector<Obstacle*>  m_Obstacles;

Container containing any walls in the environment
std::vector<Wall2D>           m_Walls;

Graph *myGraph;
std::vector<Driver*> m_Drivers;

Set true to pause the motion
  bool                          m_bPaused;
```

```
void InstanciateVehicles();
void CreateObstacles();

void CreateWalls();
```

EventData Class 22

```
 public:
  double speedx, speedy;

    double accelerationx, accelerationy;
    double angle; //relative to medial axis
     int lane_index;
```

We have not included,in this command (pData) passed onto the statemachine's functions, the index of the segment at a given intersection ,we which to turn towards, in case several possiblities exist.

VIII. Car Following Models:[4]

[4] **RICHARD W. ROTHERY, University of Texas**