Automated vehicle

Jiri Harasim

Introduction

Urban Challenge

Boss and his architecture

Trajectory generation

On-road planning

Unstructured planning

Anytime D*

Introduction

Why do we develop automated vehicle at all?

Introduction

Why do we develop automated vehicle at all?

Army usage

Handicapped people

Laziness

Why do we develop automated vehicle at all?

Army usage

Handicapped people

Laziness

In fact, for any robot it's important to be able to get somewhere safely.

Grand & Urban challenge

Grand challenge 2004

230 km in The Mojave Dessert, California

1mil USD

nobody finished

15 teams

Grand challenge 2005

211 km in The Mojave Dessert, California

2mil, 1mil, 500k

Stanley – Stanford Racing Team, 6:53

23 teams in finals – 4 more finished

Urban challenge 2007

96 km in George Air Force Base, California

each finalist received 1mil, 2mil, 1mil, 500k

Boss – Tartan Racing, 4:10

89 teams, 11 in finals – 5 more finished

Boss and his architecture

Vehicle

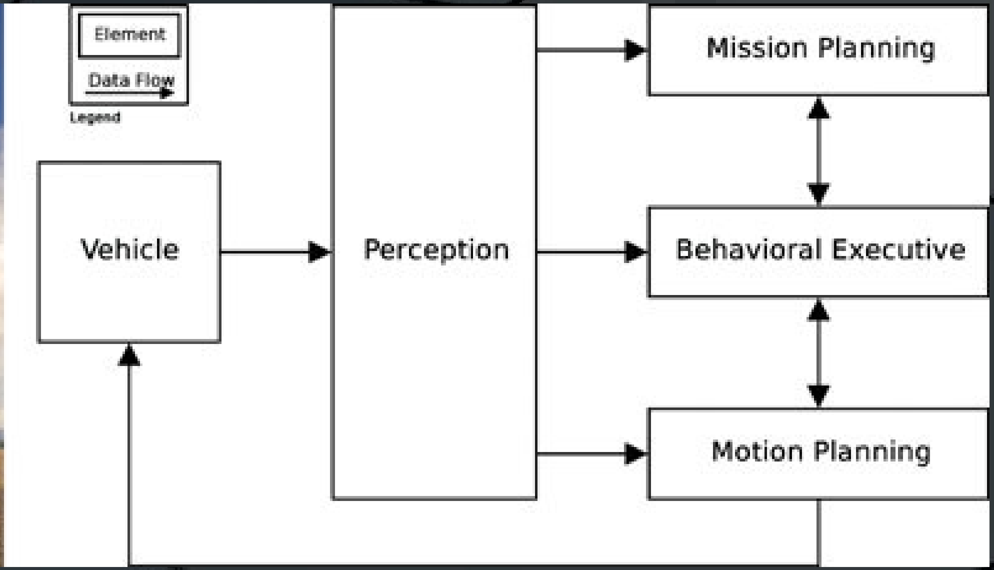
Boss is modified Chevrolet Tahoe

Has 17 different sensors

Uses GPS and sensors to localize itself in his on-board map

10x2.16GHz Core2Duo, runs Ubuntu 6.06

Has 60 parallel processes, around 500 GB HDD



“Boss” and his architecture

Perception

Vehicle state – speed, global and local position

Road world model – information about roads, intersections and parking zones

Moving obstacles – estimation of movement

Static obstacles – 2D map of free, dangerous and lethal zones

Road blockages – clearly impassable zones

“Boss” and his architecture

Mission planning

Computes the best route through the road network

Generates checkpoints

Uses gradient value function over the road network

Is easily updated

Behavioral executive

Accounts in actual traffic

Encodes rules of the roads

Implements error recovery system

“Boss” and his architecture

Motion Planning

Receives goal from behavioral executive and generates and safely executes trajectory

On-road driving

Unstructured driving

“Boss” and his architecture

World model

Includes road network, static obstacles, tracked vehicles and Boss himself

Static obstacles are represented in a regular grid of cells

Dynamic obstacles are tracked and their future position is estimated

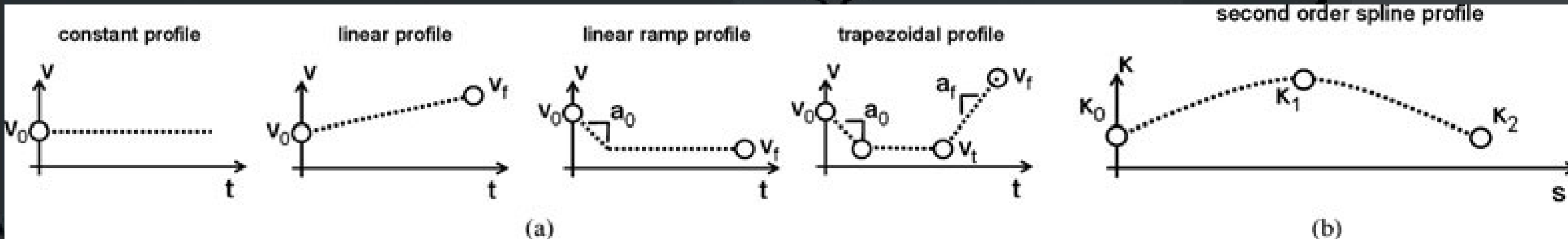
Intersections

Typical rules, as you know them

Doesn't assign precedence to specific vehicle

If nobody moves on the intersection for 10 s, Boss moves in assuming he has a precedence

Trajectory generation



Trajectory generation is a part of a motion planner

Boss uses speed profiles with respect to what his next plan is

It uses curvature profiles to approximate the shape of the road

Saved table of precomputed values for profiles, which saves large amount of time

Selects set of trajectories and works with them

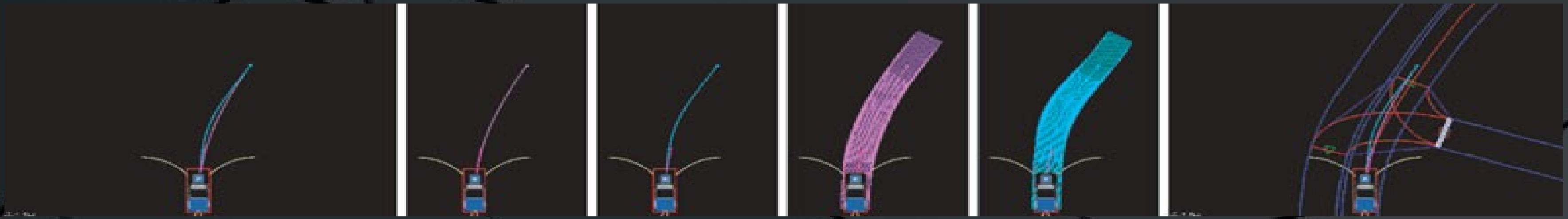
Has some level of freedom

On-road planning

Plan generation

Generates trajectory along center-line and transforms this trajectory into a set of goals to reach

For each goal uses trajectory generation described before, generating smooth and sharp trajectories



On-road planning

Velocity profiles

For every trajectory, there is a velocity profile to be chosen

Boss chooses such profile, which maximizes speed of a vehicle while being safe at the same time

Set of trajectories is evaluated against various metrics and the best one is chosen

Other moves

Line changing move

U-turn

Defensive driving

Executed by the same system, using correct goals

Unstructured planning

Plan generation

Differs a lot from on-road planning

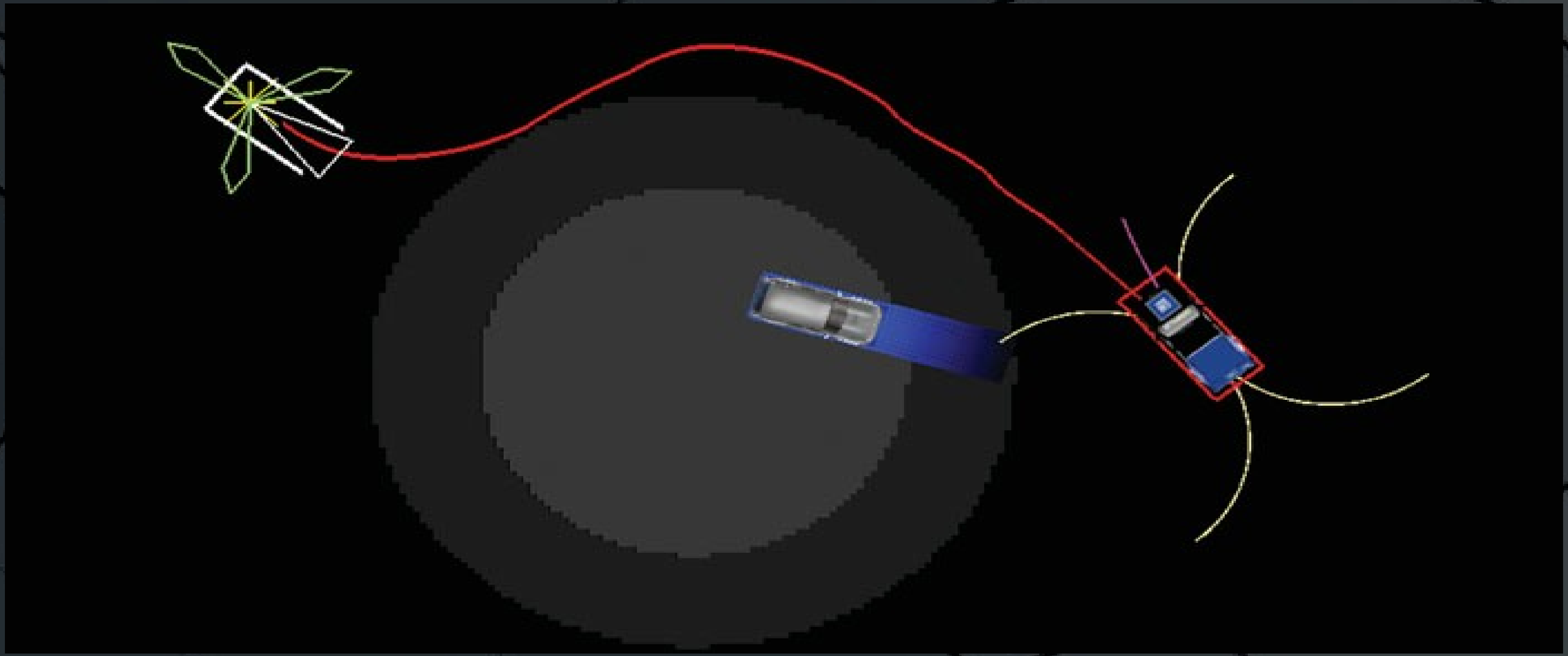
Boss uses lattice planner with (x, y, θ, v)

Boss uses Anytime Dynamic A* algorithm to find solutions to problems (Anytime D*, AD*)

Heuristic function has a huge role in this algorithm

Plan generation

Boss uses the cost map to avoid dynamic obstacles



Unstructured planning

Error recovery

Uses the same lattice planner (x, y, θ, v)

Has error level

Never gives up

Usable for blockages, jammed roads, unpredictable situations

Anytime D*

A* search

This is one of the basic search methods

Uses heuristic function to improve Dijkstra

Heuristic function h() must satisfy triangle inequality for given state *s* as follows:

$h(s_{goal},s_{goal}) = 0$ and for every $s \neq s_{goal}$, $h(s) \leq c(s,succ(s)) + h(succ(s))$

A* search

$g(s_{start}) = 0$; all other g-values are infinite; $OPEN = \{ s_{start} \}$;

While(s_{goal} is not expanded)

remove s with the smallest $[f(s) = g(s)+h(s)]$ from $OPEN$;

insert s into $CLOSED$;

for every successor s' of s such that s' not in $CLOSED$

if $g(s') > g(s) + c(s,s')$

$g(s') = g(s) + c(s,s')$;

insert s' into $OPEN$;

Anytime Repairing A*

ARA* produces a suboptimal solution very fast and gets optimum eventually

We add an ε, which we use to multiply heuristic function with it, to find suboptimal solution

We add a v() value to every state, which holds value of a state during an expansion

Anytime Repairing A*

set ε to large value;

$g(s_{start}) = 0$; *v*-values of all states are set to infinity; $OPEN = \{s_{start}\}$;

while ε ≥ 1

  CLOSED = *{}*;

  ComputePathwithReuse();

  publish current ε suboptimal solution;

  decrease ε;

  OPEN = OPEN U INCONS

ComputePathwithReuse function

While($f(s_{goal})$ > minimum f-value in *OPEN*)

remove s with the smallest $[\,g(s)+\varepsilon * h(s)\,]$ from *OPEN*;

insert s into *CLOSED*;

v(s)=g(s);

for every successor s' of s

if $g(s') > g(s) + c(s,s')$

g(s') = g(s) + c(s,s');

if s' not in *CLOSED* then insert s' into *OPEN*;

otherwise insert s' into *INCONS*

Anytime D*

D* Lite Algorithm

D* Lite produces a solution in an environment with dynamic obstacles

D* Lite stores costs from every state to goal

Stores one step look-ahead value rhs(s) which

$$rhs(s) = 0 \text{ if } s = goal$$

$$\min_{s' \text{ from } Succ(s)} (c(s, s') + g(s')) \text{ otherwise}$$

Works backwards

Isn't Anytime!

Anytime D*

```
key(s)
 return [min( g(s), rhs(s) ) + h( sstart, s); min(g(s), rhs(s))];
UpdateState(s)
 if s was not visited before
    g(s) = infinity;
 if (s != sgoal)
   rhs(s) = min s' from Succ(s) ( c( s, s' ) + g( s' ) );
 if (s from OPEN)
   remove s from OPEN;
 If ( g(s) != rhs(s) )
   insert s into OPEN with key(s);
```

Anytime D*

ComputeShortestPath()

while(min $_{s' \text{ from OPEN}}$ (key(s))<key(s_{start}) OR rhs(s_{start}) != g(s_{start}))

    remove state s with the minimum key from OPEN;

    If (g(s) > rhs(s))

        g(s) = rhs(s);

        for all s' from Pred(s) UpdateState(s');

    else

        g(s) = infinity;

        for all s' from Pred(s) [{s} UpdateState(s');

Anytime D*

Main()

```
g( sstart ) = rhs( sstart ) = infinity; g( sgoal ) = infinity;
rhs( sgoal ) = 0; OPEN = empty;
insert sgoal into OPEN with key( sgoal );
forever
   ComputeShortestPath();
   Wait for changes in edge costs;
   for all directed edges ( u, v ) with changed edge costs
      Update the edge cost c( u, v );
      UpdateState( u );
```

Anytime D*

Anytime D*

Combines both, ARA* and D* Lite into Anytime Dynamic algorithm

Has high ε, so generates suboptimal solution quickly

When change ocures, just sets high ε again

Need to consider under-consistency differently

key(s)

```
if ( g( s ) > rhs( s ) )
   return [rhs( s ) + ε * h( sstart, s ); rhs( s )];
else
   return [g( s ) + h( sstart, s ); g( s )];
```

Anytime D*

UpdateState(s)

```
if s was not visited before
   g( s ) = infinity;
If ( s != s_goal ) rhs( s ) = min_{s' from Succ(s)} ( c( s, s' ) + g( s' ) );
If ( s from OPEN ) remove s from OPEN;
If ( g( s ) != rhs( s ) )
if s ! from CLOSED
   insert s into OPEN with key( s );
else
   insert s into INCONS;
```

Anytime D*

ComputeorImprovePath()

while($\min_{s \text{ from OPEN}}$(key(s))<key($s_{start}$) OR rhs($s_{start}$) != g($s_{start}$))

  remove state s with the minimum key from OPEN;

  If (g(s) > rhs(s))

    g(s) = rhs(s);

    CLOSED = CLOSED U {s};

    for all $_{s' \text{ from } Pred(s)}$ UpdateState(s');

  else

    g(s) = 1;

    for all $_{s' \text{ from } Pred(s)}$ U {s} UpdateState(s');

Anytime D*

Main()

```
g( s_start ) = rhs( s_start ) = infinity; g( s_goal ) = infinity;
rhs( s_goal ) = 0;  ε = ε0;
OPEN = CLOSED = INCONS = empty;
insert s_goal into OPEN with key( s_goal );
ComputeorImprovePath();
publish current ε-suboptimal solution;
forever
   if changes in edge costs are detected
      for all directed edges ( u, v ) with changed edge costs
         Update the edge cost c( u, v );
         UpdateState( u );
```

Anytime D*

```
if significant edge cost changes were observed
   increase ε or replan from scratch;
else if ε > 1
   decrease ε;
Move states from INCONS into OPEN;
Update the priorities for all s from OPEN according to key( s );
CLOSED = empty;
ComputeorImprovePath();
publish current ε-suboptimal solution;
if ε = 1
   wait for changes in edge costs;
```

Anytime D*

Thank you!

References

Anytime Dynamic A*(Likhachev, Ferguson, Gordon, Stentz, Thrun)

Search-based planning with motion primitives(Likhachev)

Motion planning in urban environments(Ferguson, Howard, Likhachev)

Autonomous driving in traffic(AI magazine volume 30 number 2)

carmotor.cz(DARPA Urban Challenge 2007 – velký den pro robotiku)

wikipedia