Real-Time Motion Planning and Autonomous Driving

Jeffrey Ichnowski
What is Real-Time Motion Planning?

1st floor vs. 2nd floor

Motion planning with a hard real-time constraint.

temporally correct

\[ T_i = (\phi_i, p_i, e_i, D_i) \]
Example Real-Time Motion Planning System

$T_v$  Sense (e.g., vision)  $T_p$  Plan  $T_c$  Move (controller)
Problems with Motion Planning in a Real-Time System

- Motion planning is P-SPACE hard.
- Exponential in dimensions.
- Uncountably infinite state spaces.

[Image source: wikipedia]
Real-Time Motion Planning

In the general case: impossible.
Precompute Motion Plan

Extremely popular option.

- Allows arbitrarily long computation
- Asymptotically feasible algorithms
- Asymptotically optimal algorithms

Is this real-time? yes and no.
Real-time Motion Planning Problem

Time-bounded computation

Responsive a dynamic environment
(moving obstacles, goals, new data)
Collision Avoidance

- Reformulation of path planning into collision avoidance.
- Potential fields
- Real-time ✓
- Problem: gets stuck in local minima

[Khatib 1986]
Anytime Planners

- Incrementally build a planning tree
- May be stopped at **anytime**
- Example: RRT, RRT*
- Real-time?
- Reactivity: rapid re-planning + some luck

[Image source: Ichnowski 2013]
Roadmap Planners

• Pre-computes a roadmap (connectivity of freespace)

• Motion plan = graph search

• Real-time?

• Example: PRM

• Reactivity must come from another task.

[Image source: LaValle 2006]
Grid/Lattice Planners

1. Discretize space
2. Plan in the discretized space

Often done with A* or variant.

Weighted A*: reduced plan optimality & compute time

D* is reverse A* + keep data for next compute cycle

[Image source: Pivtoraiko 2006]
A*  

- Provably optimal  
- What if graph changes during the search? (e.g., dynamic environment)  
- $O(b^d)$, $d =$ solution length, $b =$ branching factor. (polynomial if search space is tree*)
Real-Time Heuristic Search

Minimin + $\alpha$-Pruning

1. Depth-limited horizon search
2. A* metric frontier ($S$)
   \[ f(x) = g(x) + h(x) \]
3. Take step towards best frontier node
4. repeat.

Assign $\alpha = \min_{x \in S} f(x)$

Prune search when $f(x) \geq \alpha$

[Korf 1988]
Real-Time A*

- Use Minimin w/ $\alpha$-Pruning in “planning mode”
- RTA* used in execution.

RTA*: At $x_i$, what is $x_{i+1}$?

1. Choose $x_{i+1} = \arg\min_{x' \in \text{neighbors of } x_i} g(x') + h(x')$
2. Store **second** best $g(x') + h(x')$ for $x_i$

[Korf 1988]
Partitioned-Learning RTA*

- Start w/ RTA* (depth-limited search)
- Take step towards best path
- “Learn” $h(x)$ of all frontier
- Split $f(x)$ into dynamic and static components

\[
f(x) = g_s(x) + g_d(x) + h_s(x) + h_d(x)
\]

[Cannon et. al. 2014]
Real-Time $R^*$ (RTR*)

$R^* \approx RRT + A^*$

**RTR**

- fixed # of node expansions
- choose best frontier node (path and min $g(x) + h(x)$)
- geometric expansion limits for difficult nodes
- path reuse

[Cannon et. al. 2014]
Hard-Real-Time Rapidly-exploring Randomized Trees

procedure HRT_PLANNER
  t_next = current_time()
loop
  yield until t_next
  t_next = t_next + T_p
  B = updated map
  q_init = current vehicle state
  q_goal = current goal states
  T = BUILD_RRT(q_init, q_goal, n)
  path = EXTRACT_PATH(T)
publish path

Figure 5.5: A demonstration of the approach used for extending the RRT search trees

The size of the search tree is the factor which will have the most significant effect on execution time.

This formulation makes a number of simplifying assumptions. Primarily, the time required to extend the search tree must be constant. This in turn requires that only a constant number of collision detection tests. For sampling based collision detection strategies this can be achieved by limiting the time step used for each tree extension.

The number of nodes added to the search tree RRT is a measure of search effort. If the search is terminated before a solution is found, the problem is considered to be infeasible. Consequently, the task of finding an efficient hard real-time equivalent to RRT can be addressed by finding the relationship between $n$ and the WCET.

5.2 Execution Time Estimation

Selection of an appropriate technique for execution time measurement for hard real-time planners depends on a number factors including the required accuracy and the target hardware. Many of the techniques presented in the catalogue of execution time estimation methods in Section 3.4 could be used to solve this problem. For the purposes of verification of the RT- execution period

solution or "safe" path

number of samples (WCET analysis)

[Walker, 2011]
WCET Estimation Tool-chain

A critical aspect of the validation of the execution time estimation process is to ensure that all branches in the program have been adequately measured. This can be achieved by performing either coverage testing, or by comparing a representation of the observed control flow graph (CFG) with one produced directly from the disassembled object code of the planner. These processes can then be used to guide the introduction of additional instrumentation statements, or to selectively re-execute the planner in different initial conditions so as to improve the coverage of execution traces.

[Walker, 2011]
The results of the execution time profile of the RRT planner running in the infeasible workspace are shown in Figure 5.16. This demonstrates a clear difference to the empty workspace. This occurs because it is not possible to find a path to the goal, the planner will continue searching until the available storage resources are exhausted. This results in the observed execution times being tightly clustered just below the theoretical WCET curve.

The results for the obstructed workspace are shown in Figure 5.17. They show that the distribution of observed execution times for a problem that is more difficult than the empty workspace, and easier to solve than the infeasible case. The observed execution times, regardless of search tree size, are more skewed toward the WCET curve than for the empty workspace.

Significantly, this result also highlights the clear dependency between search tree size, and the amount of execution required to find a solution in the worst case. By considering a series of planning problems with varying obstacle placement demonstrated that there is also a correlation between the difficulty of a planning problem and how much time is required to find a solution.

The locations of obstacles within the workspace are a factor which can have a significant influence on the execution path followed by the RRT algorithm. This observation can be exploited to ensure that the coverage of instrumentation is adequate. To achieve good measurement coverage three basic workspaces were considered: the empty environment (with no obstacles); the infeasible workspace (with a single obstacle obstructing any path to the goal); and a further obstructed workspace, designed to be feasible, but more difficult than the empty workspace.

The empty, infeasible and obstructed workspaces, along with sample search trees produced using RRT in each are shown in Figures 5.9, 5.10 and 5.11. The boundaries of these images denote the shape of the workspace, the dark grey circles denote the obstacles, the red circles are the location of the goal configurations, and black lines are indicative of the search trees produced by the RRT algorithm during the experiments.
The locations of obstacles within the workspace are a factor which can have a significant influence on the execution path followed by the RRT algorithm. This observation can be exploited to ensure that the coverage of instrumentation is adequate. To achieve good measurement coverage three basic workspaces were considered: the empty environment (with no obstacles); the infeasible workspace (with a single obstacle obstructing any path to the goal); and a further obstructed workspace, designed to be feasible, but more difficult than the empty workspace.

The empty, infeasible and obstructed workspaces, along with sample search trees produced using RRT in each are shown in Figures 5.9, 5.10 and 5.11. The boundaries of these images denote the shape of the workspace, the dark grey circles denote the obstacles, the red circles are the location of the goal configurations, and black lines are indicative of the search trees produced by the RRT algorithm during the experiments.

Figure 5.9: The Empty Workspace  
Figure 5.10: The Infeasible Workspace  
Figure 5.11: The Obstructed Workspace

[Walker, 2011]
Obstructed Workspace

The empty workspace exercises the branches in the RRT algorithm that deal with successfully finding the goal. This means that only the constraints on the motion of the vehicle can prevent a solution from being found. In a workspace of this shape, this environment is effectively the best possible case.

The infeasible workspace is an example of a problem that is close to the pathological worst-case. The presence of an obstacle preventing access to the goal results in the search tree expanding, without early termination because a goal is found. This case is expected to result in near worst-case performance.

The obstructed workspace was selected to represent a more typical case. The narrow passages around the obstacle result in an interleaving of expansions, and collisions that would indicate whether any micro-architectural effects were having a significant contribution to the final result.

Within each of the workspaces, 100 trials were conducted to produce a composite execution trace. Additional trials could be conducted to improve the quality of the execution time fits for each edge in the CFG, but these values tended to converge with many fewer than 100 trials. Similarly, a much smaller number of trials could have been used to identify all of the edges within the CFG.

An example of a subset of the Control Flow Graph produced by analysing the combined execution trace, with calls to functions omitted, is shown in Figure 5.12. This highlights that the key branches within the RRT algorithm all appear within the CFG. For example, the cycle of vertices 8, 9, 3, 4 and 7 are a result of the main loop within the RRT, and the branch between 9 and 10 is caused by the early exit of the planner when a solution has been found.

In order to ensure that an edge is measuring an approximately constant time operation, it is important to note that the labels on the CFG in Figure 5.12 are a function of the instrumentation process, and are only shown to assist in describing parts of the algorithm.
6.2.3 Slack Time

The aim when designing a real-time system is typically to ensure that a particular combination of tasks are schedulable. In the case where the execution time, and quality of results, of a task can be modified by a parameters the problem of real-time scheduling becomes one of

[Walker, 2011]
Real-Time Motion Planning for Autonomous Vehicles

Reduce dimensionality of planning problem. Typically around 4: e.g., \((x, y, \theta, v)\)

Discretization and sampling-based approaches
DARPA Urban Challenge 1st place: Tartan Racing (CMU)

local planner at 10 Hz fixed

lattice planner at 10 Hz (nominally)

- Difficult scenarios take “up to a couple of seconds” (motivation for their pre-planning)

- Anytime planner example: first solution in 100 ms, optimal at 650 ms.

- Time for replanning “few ms” for small adjustments to “few seconds” for drastically different trajectories

[Likhachev 2008] & [Ferguson 2008]
DARPA Urban Challenge 1st place: Tartan Racing (CMU)

Anytime planner behavior

solution found <100 ms

optimal solution < 650 ms

solution improved

[Likhachev, et. al. 2008]
DARPA Urban Challenge 1st place: Tartan Racing (CMU)

Effect of heuristic on A* search

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>States Expanded</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h$</td>
<td>2,019</td>
<td>0.06</td>
</tr>
<tr>
<td>$h_{2D}$</td>
<td>26,108</td>
<td>1.30</td>
</tr>
<tr>
<td>$h_{fsh}$</td>
<td>124,794</td>
<td>3.49</td>
</tr>
</tbody>
</table>

Implies much higher WCET

[Likhachev, et. al. 2008]
“One of the important lessons learned during the development of this system was that it is often extremely beneficial to exploit prior, offline processing to provide efficient online planning performance.”

– Ferguson, Howard, and Likhachev
DARPA Urban Challenge 2nd place: Stanford Racing

- Sensors at 10 Hz
- RNDF\(^1\) editor at 10 Hz
- Full replanning: 50 to 300 ms
  1. hybrid A\(^*\) (unnatural swerves)
  2. conjugate-gradient descent smooth (0.5 m)
  3. interpolation (5 to 10 cm)

\(^1\) route network definitions file

[Montemerlo et. al. 2009] [Dolgov et. al. 2010]
**Fig. 5.** Hybrid-A* and CG-smoothed paths for a complicated maneuver, involving backing out of and into parking spots. The Hybrid-state A* path is the wavy red line and the CG solution is the smooth blue line.

**Fig. 6.** Anchoring waypoints to guarantee smoother safety.

The reason is that the potential attempts (within its effective range) to maximize the distance between every vertex of the trajectory and the nearby obstacles. However, this is not always the right solution. For example, when approaching a narrow gap between obstacles at an angle (as illustrated in Figure 6), the trajectory for the center of the rear axle of the vehicle stays closer to one side of the gap, allowing the car to safely turn into the gap. Unfortunately, because the collision-potential used in the smoother does not model the shape of the vehicle, it is unable to do such precise collision detection and will center the trajectory within the gap, resulting in an unsafe maneuver.

While it is possible to model precise collision detection within the smoother, it is computationally prohibitive. In particular, computing the derivative of the collision cost with respect to the coordinates of the path of the rear axle is a computationally intensive task which has to be performed within the inner loop of CG optimization.

Therefore, we opt for a computationally simpler (albeit less elegant) solution that guarantees that smoother output is collision-free. We use an iterative approach that works, as follows. We run the CG smoother and check its output for collisions. If we find any unsafe states, we anchor them to the A* solution (the smoother is not allowed to modify the coordinates of anchored states) and re-run the smoother. This process is repeated until the smoother output is collision-free. It is guaranteed to converge because A* path is guaranteed to be safe. In the worst case, the smoother will return the same solution as A*, which will only happen under extreme circumstances.

Figure 6 illustrates the process. As before, the wavy red curve shows the path produced by A*, while the straight blue line is the output of the smoother. The circles designate states that ended up being anchored to the A* path (i.e., not allowed to move). Notice that in this rather constrained problem, only a few states are locked down, while the trajectory is successfully smoothed.

A video corresponding to the situation in Figure 6 is available from the following URL: http://ai.stanford.edu/~ddolgov/gpp/anchors.avi.

**3.3. Navigation Potential Using the Voronoi Field**

One issue that we have omitted from our discussion of path planning so far is the trade-off between proximity to obstacles and trajectory length. A weakness of the path-planning algorithm as described in the previous sections is that it tends to “hug walls”, i.e., it will choose the minimal-length trajectory that is collision free, often causing the robot to drive at the minimal collision-free distance to obstacles.

A common way of penalizing proximity to obstacles is to use a potential field (see, e.g., Andrews and Hogan (1983), Pavlov and Voronin (1984), Miyazaki and Arimoto (1985) and Khatib (1986)). However, conventional potential fields have a couple of important drawbacks. First, as has been observed by many researchers (see, e.g., Tilove (1990) and Koren and Borenstein (1991)), conventional potential fields create high-potential areas in narrow passages, which can make the cost of traversing these passages prohibitively high. Second, which plays an even more important role in our approach, is computational efficiency. A straightforward potential around an obstacle is typically defined as a function of the distance to the obstacle. This means that in order to compute the value of such a potential at a given point, we need to compute the contributions of the potentials from all obstacles that contain within their effective radius. This can be computationally expensive. A common technique to avoid this issue is to approximate the potential by using only the contribution of the potential from the nearest obstacle, which can be computed much more effectively. However, this introduces another problem. Since we will use the potential within the CG smoother, we need the potential to be smooth and have a well-defined derivative.
DARPA Urban Challenge 2nd place: Stanford Racing

Fig. 13. Left: Trajectory driven in simulation using the free-space version of our planner. The robot had to replan in response to obstacles being detected by its sensors. This explains the apparent sub-optimality of the trajectory. Right: Replanning times for the maze-like environment (total time, A* time, smoothing time.)

Figure 12(i) is interesting because Junior had to navigate around other cars near the entrance into the zone. A video of the parking task in Figure 12(a) is available at http://ai.stanford.edu/~ddolgov/gpp/duc_nqe_park.mpg.

Figures 12(c)–(f) show U-turns on blocked roads that were performed using the free-space planner. Videos of Junior performing U-turns are available at http://ai.stanford.edu/~ddolgov/gpp/duc_nqe_uturn.mpg and http://ai.stanford.edu/~ddolgov/gpp/duc_nqe_uturn2.mpg.

Figure 12(h) shows a parking task during the DUC race. The maneuver was straightforward, because there were no obstacles in the parking lot. After parking in the designated spot, in accordance with the DUC rules, Junior backed out of the spot before proceeding to the parking-lot exit.

Figure 12(i) shows the start of one of the missions during the DUC race. Each DUC mission started with a free-navigation zone, which was traversed using the free-space planner described in this paper.

Most of the path-planning tasks in the DUC were fairly simple. As an example of the performance of our free-space planner in a more complex environment, consider the trajectory shown in Figure 13. This example was generated in simulation, the simulated vehicle was equipped with a single planar laser range finder mounted on the front of the car. Such intentionally poor (simulated) sensing led to frequent replanning as obstacles were incrementally detected. This is the source of the apparent sub-optimality of the path shown in Figure 13. A video showing the robot driving through the environment and replanning as it detects new obstacles and builds an obstacle map in scenario of Figure 13 is available at http://ai.stanford.edu/~ddolgov/gpp/maze.mpg.

Figure 14 illustrates the benefits of using the Analytic Reed–Shepp expansions described in Section 2.2. The graph shows re-planning time for a representative run in a parking lot with and without the Reed–Shepp expansions. The units are relative time normalized by the average planning time when using Reed–Shepp expansions. As was mentioned earlier in Section 2.2, Reed–Shepp expansions are not strictly guaranteed to improve planning time (because of the constant-time overhead), but in practice lead to noticeable efficiency gains.
DARPA Urban Challenge 4th place: MIT

- Drivability map updated 10 Hz
- Controller ran at 25 Hz
- RRT at 10 Hz
  - 700 samples per second

[Kuwata, et. al. 2009]
procedure RRT_execution_loop
repeat
  update vehicle states and env
while \( t < t_0 + \Delta t \)
  EXPAND_RRT_TREE()
repeat {
    \( \tau = \text{EXTRACT\_BEST\_SAFE\_PATH()} \)
    if NO safe path
      E-STOP! & restart
  }
until \( (\text{clear}(x)) \forall x \in \tau \)
send \( \tau \) to controller

[Kuwata, et. al. 2009]
B. Race Results

The following subsections present results from the National Qualification Event (NQE) and the final Urban Challenge Event (UCE). NQE consisted of three test areas A, B, and C, each focusing on testing different capabilities. During NQE, the CL-RRT algorithm was not tuned to any specific test area, showing the generality of the approach. UCE consisted of three missions, with a total length of about 60 miles. Talos was one of the six vehicles that completed all missions, finishing in 5 hours 35 minutes.

1) High speed behavior on a curvy road: Figure 7 shows a snapshot of the environment and the plan during UCE. The vehicle is in the lower left, going towards a goal in the upper middle of the figure. The small green squares represent the safe stopping nodes in the tree. The vehicle is moving at 10 m/s, so there are no stopping nodes in the close range. However, the planner ensures there are numerous stopping points on the way to the goal, should intermittently detected curbs or vehicles appear. Observe that even though the controller inputs are randomly generated to build the tree, the resulting trajectories naturally follow the curvy road. This road segment is about 0.5 mile long, and the speed limit specified by DARPA was 25 mph. Figure 8 shows the speed profile and the lateral prediction error for this segment. Talos reached the maximum speed several times on straight segments, while slowing down on curvy roads to observe the maximum lateral acceleration constraints. The prediction versus execution error has the mean, maximum, and standard deviation of 0.11 m, 0.42 m, and 0.093 m respectively. Note from the plot that the prediction error has a constant offset of about 11 cm, making the maximum error much larger than the standard deviation. This is due to the fact that the steering wheel was not perfectly centered and the pure-pursuit algorithm does not have any integral action to remove the steady state error.

Note that when the prediction error happens to become large, the planner does not explicitly minimize it. This occurs because the vehicle keeps executing the same plan as long as the repropagated trajectory is feasible. In such a scenario, the prediction error could grow momentarily. For example, during a turn with a maximum steering angle, a small difference between the predicted initial heading and the actual heading can lead to a relatively large error as the vehicle turns. Even with a large mismatch, however, the repropagation process in Section V-C ensures the safety of the future path from vehicle’s current state.

2) Unsafe Nodes in the Dynamic Environment: Figure 9 is a snapshot from the merging test in NQE. Talos is in the bottom of the figure, trying to turn left into the lane and merge into the traffic. The red lines originating from Talos show the unsafe trajectories, which do not end in a stopped state. Before the traffic vehicle comes close to the intersection, there were many trajectories that reach the goal. However, as the traffic vehicle (marked with a green rectangle) approached, its path propagated into the future blocks the goal of Talos, as shown in Figure 9, which rendered the end parts of these trajectories infeasible. However, the feasible portion of these trajectories remain in the tree as unsafe nodes.

DARPA Urban Challenge 4th place: MIT

Lane following on a curve at 22.4 mph. The green dots are safe stopping nodes. [Kuwata, et. al. 2009]
DNF. Froze at entrance to traffic circle (who doesn’t their first time?)

Software exception during mode switch
Caught by error handler, and left hanging
Not observable by watchdog module.

One of the few cars that drove collision-free

One of the authors Matthias Goebl from Institute for Real-Time Computer Systems, Technical University of Munich

[Kammel, et. al. 2008]
Multi-level control:
A. Mission planning
B. Maneuver planning
C. Collision avoidance
D. Reactive layer
E. Vehicle control

Motion planning on discretized grid of 3D configuration space using A*.

Convolutional filters used to precompute free C-space.

DARPA Urban Challenge finalist: AnnieWay (KIT)

[Kammel, et. al. 2008]
Conclusions

- Real-time motion planning difficult
- No guarantees on solution
- Multiple levels of planning
- Time-bounded computation
- Generate “safe routes”
- Keep around information between task cycles
Thank you