Overhead Optimization for Trajectory Planner

Will Baker // Christopher Zakian **UROC: Team Auto Frogger** Mentor Jeff Johnson // Advisor Kris Hauser

The trajectory planner is designed to be used in real-time simulations systems to test collision avoidance algorithms.

The planner constructs a safe, time-optimal trajectory across an intersection in the presence of oncoming vehicles (referred to here as "obstacles") [1]. It produces a set of controls that guide the vehicle across the intersection safely. Collision avoidance algorithms can thus use the planner to steer a vehicle out of collision.

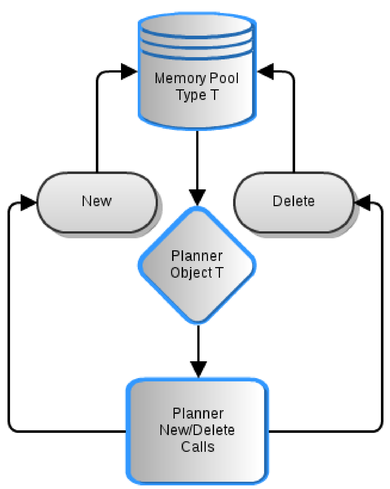
Objective

To reduce overhead required for running the algorithm. We incorporate a fixed-block-size allocation (memory pool) scheme into the trajectory planner in order to optimize memory allocation and deallocation operations. In addition, we refactor STL vector operations to remove unnecessary bounds checking.

Implementation

- ***C++ STL vector refactoring*:** We refactor vectors to use bracket accessors instead of the .at() method to remove unnecessary bounds checking.

- ***Memory Pool*:** We incorporate a memory pool for allocation and deallocation of C++ objects.



- The Planner's calls to new and delete in C++ have been overidden to be calls to a static memory pool of type T for each object.

- A call to new returns a pointer to a chunk of memory of size T.

- A call to delete marks the address of size T as free for assignment.

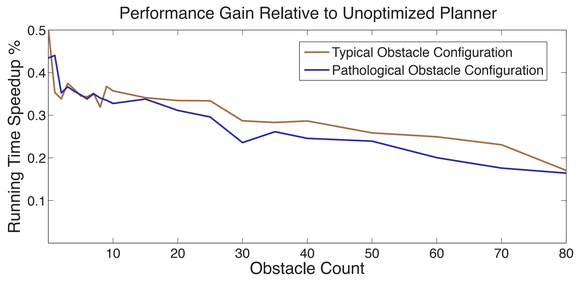


Figure 1. The typical obstacle configuration represents a configuration likely to be encountered in practice. The pathological configuration is one specifically constructed to bog down the planner. This graph displays relative performance improvement for scenarios of both configurations as obstacle count increases.

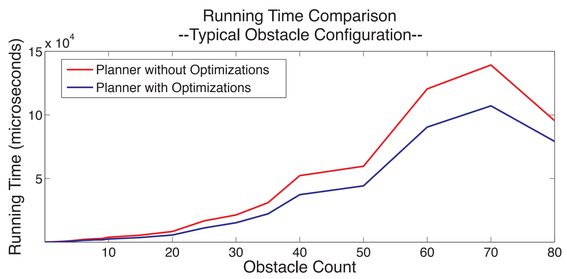


Figure 2. This graph shows absolute performance improvement for the typical configuration as the obstacle number increases.

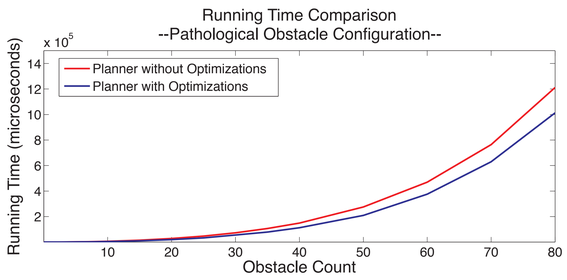
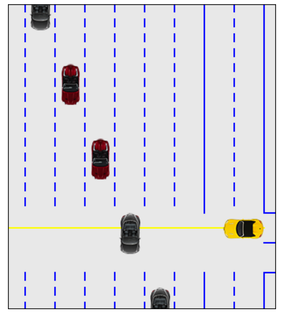
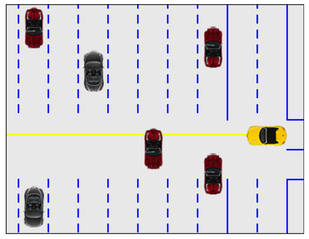


Figure 3. This graph shows absolute performance improvement for the pathological configuration as the obstacle number increases.





***Above*:** A typical obstacle configuration
***Left*:** A pathological configuration.

Results & Analysis

- Using the memory pool and refactored vector operations we have a faster and more consistent performance with the trajectory planner.

- In figure 1, as the number of obstacles increase, the ratio of time spent in computation of the trajectories grows, while the amount of overhead stays relatively constant.

- Overall there is a decrease in runtime with our optimizations (Figure 2 & 3) because we recycle the program's objects that would otherwise be freed and reallocated.

Conclusions & Future Work

Any collision avoidance system needs to operate in real time. Using the memory pool and refactored vector operations we have faster and more consistent performance with the trajectory planner.

Further performance gains may be had by more targeted optimizations. One avenue of investigation would be to profile the algorithm to identify bottlenecks that may be candidates for further optimization.

References

1. J. Johnson and K. Hauser, "Optimal acceleration-bounded trajectory planning in dynamic environments along a specified path," in International Conference on Robotics and Automation (ICRA), St. Paul, USA, May 2012.

INDIANA UNIVERSITY
BLOOMINGTON