ARS ADVANCED ROBOTIC SYSTEMS INTERNATIONAL

A Parallel Randomized Path Planner for Robot Navigation

S. Sengupta
Department of Aerospace Engineering,
IIT Kharagpur,Kharagpur-721302.
India.
shanks.iit@gmail.com

Abstract: *Computation of a collision-free path for a movable object among obstacles is an important problem in the fields of robotics, CIM and AI. Various automatic task level programming systems can be build for robot guidance, teleoperation, assembly and disassembly among others, if a suitable method for motion planning is available. In the basic variation of motion planning, the task is to generate a collision-free path for a movable object among known and static obstacles. Classically the problem was defined for a rigid 6 degrees-of-freedom body as 'the piano mover's problem'. However, the majority of the research has been conducted in the field of robotics, often under the title of path planning. Rapidly-Exploring Random Trees (RRTs) are a recently developed representation on which fast continuous domain path planners can be based. In this work, we have built a parallel path planning system based on RRTs that interleaves planning and execution, first evaluating it in simulation and then applying it to physical robots. Our distributed algorithm, PRRT (parallel RRT), introduces a parallel extension of previous RRT work, the process splitting and parallel cost penalty search with a comment on Real Time Stagnancy reduction, which improves re-planning efficiency, decreases latency involved in finding feasible paths and the quality of generated paths. PRRT is successfully applied to a real-time multi-robot system. In this paper we illustrate how it is possible to implement a parallel version of RRT based motion planner which yields optimal speed up.*
Keywords: *Motion planning, parallel systems, scalability, efficiency, process splitting*

1. Introduction

Path planning has been a much studied problem over the past two decades, whose appeal stems from its applicability to many diverse areas spanning industrial robot locomotion, autonomous actors in computer animation, protein folding, drug design etc.Specifically, in complicated, fast evolving environments such as Robocup (Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E., 1995), currently popular approaches to path-planning have their strengths, but still leave much to be desired. In particular, most require a state discretization and are best suited for domains with relaxed time constraints for planning. One of the recently developed tools that may help tackle the problem of real-time path planning are Rapidly-exploring random trees (RRTs) (LaValle,.M.,1998).RRTs employ randomization to explore large state spaces efficiently, and can develop the basis for a probabilistically complete though non-optimal Kino dynamic path planner (LaValle,S.M. and Kuffner,J., 2001). Their strengths are that they can efficiently find paths in high dimensional spaces because they avoid the state explosion that discretizes faces. Furthermore, due to their incremental nature, they can maintain complicated kinematical constraints if necessary. A basic planning algorithm using RRTs is as follows: Start with a trivial tree consisting only of the initial configuration. Then iterate: With probability p, find the nearest point in the current tree and extend it towards the goal G. Extending means adding a new point to the tree that extends from a point in the tree toward G while maintaining whatever kinematical constraints exist. In the other branch, with probability 1 − p, select a point X uniformly from the configuration space, find the nearest point in the current tree, and extend it toward X. Thus the tree is built up with a combination of random exploration and biased motion towards the goal configuration. Most current robot systems that have been developed to date are controlled by heuristic or potential field methods at the lowest level, and many extend this upward to the level of path navigation (Latombe, C., 1991).

Since the time to respond must be bounded, reactive methods are used to build constant or bounded time heuristics for making progress toward the goal. One set of reactive methods that have proved quite popular are potential fields and motor schemas (Arkin, R.C., 1989). Although they meet the need for action under time constraints, these methods suffer from the lack of look ahead, which can lead to highly non-optimal paths and problems with oscillation. This is commonly accepted, and dealt with at a higher layer of the system that detects failure or a local minimum and tries to break out of it.

RRTs, as used in our work and presented in this paper, should provide a good compliment for very simple control heuristics, and take much of the complexity out of composing them to form a navigation system. Specifically, local minima can be reduced substantially through look ahead, and rare cases need not be enumerated since the planner has a nonzero probability of finding a solution on its own through search. Furthermore, a parallel RRT system can be fast enough to satisfy the tight timing requirements needed for fast navigation.

While not as popular as heuristic methods, non-reactive planning methods for interleaved planning and execution have been developed, with some promising results. Among these are agent-centered A* search methods (Koenig, S., 2002) and the D* variant of A* search (Stentz, A., 1995). However, using these planners requires discretization or tiling of the world in order to operate in continuous domains. This leads to a tradeoff between a higher resolution, with is higher memory and time requirements, and a low resolution with non-optimality due to discretization. Most of the features of agent-centered search methods do not rely on A* as a basis, however, so we can achieve many of their benefits using an RRT based planner which fits more naturally into domains with continuous state spaces. The distributed RRT planner we developed is roughly competitive with these other methods in that both can meet tight timing requirements and can reuse information from previous plans, but at this point it does not perform significantly as expected with the parallel paradigms being implanted although the base RRT system is relatively easy to extend to environments with moving obstacles, higher dimensional state spaces, and kinematics constraints. The primary goal of our work was thus to demonstrate the feasibility of an RRT-based algorithm with a parallel approach on a real robot planning at real-time rates. Results demonstrate that PRRT is significantly more efficient for re-planning than a basic RRT planner, performing competitively with or better than existent heuristic and reactive real-time path planning approaches. PRRT is a significant step forward with the potential for making path planning common on real robots, even in challenging continuous, highly dynamic domains using parallel computation techniques with considerable advantage for taking real time decisions.In the last few years a number of new application fields requiring the solution of problems involving not only state space constraints but also differential constraints have been introduced. The so called Kino dynamic motion planning problem (Donald, B., Xavier, P., Canny, J. and Reif, J., 1993) has to be solved when designing digital actors, humanoid robots, virtual prototyping systems, or to study molecular structures (Latome, J.C., 1999). Since even the simpler generalized mover's problem is PSPACE-hard (Reif, J.H., 1979), approximated and random techniques have been introduced (Kavraki, L.E., ˇSvestka, P., Latombe, J.C., and Overmars, M.H., 1998). Probabilistic path planners achieve probabilistic completeness, that is, provided that a solution exists, by allotting more time to the planner we can improve the chance it will eventually find a solution. They are then being used to solve Kino dynamic problems characterized by high dimensional configuration spaces. In this scenario, Rapidly-exploring Random Trees (RRT) (LaValle,S.M.,1998) exhibit good results and have been applied for solving real world problems about systems involving multi degrees of freedom (Kuffner,J., Nishiwaki,K., Kagami,S. , Inaba,M. I. and Inoue, H., 2001). Nevertheless searching a path in a high dimensional configuration space is still a time consuming task and the only way to reduce it is by using effective parallel and distributed techniques.

2. The RRT Algorithm

The problem statement is the following: given a metric space X, a starting point Xinit ∈ X and goal regions Xgoal ∈ X or goal state Xgoal ∈ X, find a continuous path from Xinit to Xgoal which does not intersect the region Xobs ∈ X. In a classical motion planning problem the metric space X is the configuration space of the robot. In a Kino dynamic motion planning problem X is constrained to be a subset of the tangent bundle of the configuration space ($X \in T(C)$). We briefly review RRT-connect (Reif, J.H., 1979), one of the basic versions of RRT. As illustrated in algorithms 1.(a) and 1.(b), a tree is incrementally built, starting from the initial configuration. Here we show a different approach but the underlying principle of RRT construction remains the same at the lowest level (LaValle, S.M., 1998). Each tree node is associated with a configuration satisfying the dynamic constraints of the system. The search ends when the tree reaches Xgoal or when the allotted time expires (thus to prevent infinite search over an unsolvable problem instance). During the ADD step a new node tree is added by integrating the differential constraints of the system (NEW CONFIG) and checking its status and location in Xfree. The new configuration is created starting from a randomly generated configuration and the closest configuration in the tree being grown. The strength of this approach is that with this expansion technique, the tree is biased to grow towards unexplored regions of the free configuration space (Xfree). Another advantage is that the path being built satisfies the dynamical constraints of the systems, thus avoiding the classical two steps approach involving path search and subsequent smoothing. An analysis of the asymptotic behavior of the RRT algorithm illustrates that the distribution of the samples converges to the random sampling process used to get the samples. It can also be shown that the RRT-Connect algorithm is probabilistic complete (Reif, J.H., 1979). It is clear that the size of the output produced, i.e. the number of nodes in the solution tree, is a random variable itself.

```
RRT PLANNER (Xinit, Xgoal,K)
1  INPUT starting and goal points xinit and xgoal;
2   _a;Init(Xinit);
4   _b;Init(Xgoal);
5   for k = 1 to K do
6       Xrand ← RANDOM CONFIG( );
7       if NOT ADD (_a;qrand) = Trapped then
8          if ADD (_b; qnew) = Reached then
9             RETURN PATH (_a; _b);
10         End if
11      End if
12      SWAP (_a; _b);
13  End for
14  RETURN Failure;
```

1(a). RRT pseudocode for the RRT Planner.

```
ADD (; q)
1   INPUT a tree and a random configuration q
2   RETURN Trapped or Reached or Advanced
3    qnear ← NEARESTNEIGHBOR (T, Xrand)
4    if NEW CONFIG (q; qnear; qnew) then
5         _.add vertex (qnew)
6         _.add edge (qnear; qnew)
7         if qnew = q then
8            RETURN Reached
9         else
10         RETURN Advanced
11        End if
12    End if
13  RETURN Trapped
```

1(b). ADD vertex pseudocode for the RRT Planner.

3. Parallel Formulations of the RRT Algorithm

One of the possible ways to improve the performance of the RRT algorithm is to develop a parallel implementation. Indeed *parallel motion planning algorithms* have been already studied and it has been shown that this can be a viable opportunity to speed up paths computation (Henrich, D., 1996). In the framework of *randomized* algorithms, two major ways can be undertaken. The first approach is based on the *parallel process splitting paradigm* approach, where a set of processors is engaged in the solution of the same instance but different data paths or division of processes occurs, like in MIMD computers, of the search problem. Each processor executes the same algorithm and the first one which finds a solution sends a message to the others to stop their computation (Caselli,S. and Reggiani,M.,2000). In this way it is not necessary to reformulate the algorithm, but it suffices to just add a few communication steps.

3.1. Process Splitting Parallel RRT Implementation

In the so called *PROCESS SPLITTING parallel* paradigm a set of processors solves the same computational problem using the same randomized algorithm. It is also possible to let all the processors terminate the computation and then take the best one as the solution produced by the set according to some quality index (Atramentov,A. and LaValle,S. M.,2002), but we will not consider this variant here. In the basic approach of parallel computation every processor then carries out all the computation on its own and communication is performed just when the first processor finds a solution. Then a *termination message* is broadcasted to all the other processors to let them terminate (for this reason this method is also called *barrier parallel* because the first which finds a solution stops all the others). The goal of this method is to minimize the time needed to compute a solution. The theoretical explanation (Challou, D., Boley,D., Gini,M.,Kumar,V., and Olsen,C. ,2002) stems from the observation that the time required to find a solution to the motion planning problem using the RRT algorithm is a random variable. Let us suppose that m processors execute the PRRT algorithm to solve the same problem instance and let Ti be the time spent by the i-th processor.

Pi (t) = probability of existence of feasible path

All the Tis are independent and identically distributed. Let Pi (t) be the probability that the time spent by processor i to solve the problem will not exceed t, i.e.

$$\mathbf{Pi\ (t) = Pr\ [Ti < t]} \qquad (1)$$

Thus the probability Pm~(t) that none of the m processors will find a solution in time t is Pm none(t) which is

$$\mathbf{Pm\sim\ (t) = \Pi\ (1 - Pi\ (t)),i=1(\ )K} \qquad (2)$$

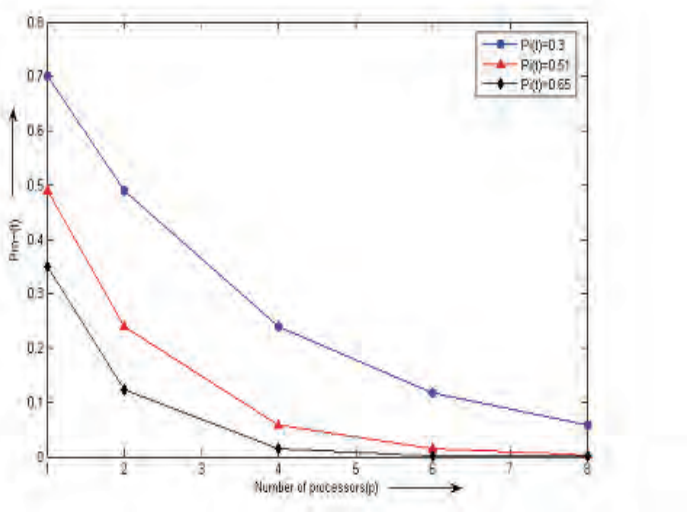


Fig. 1. Figure illustrates the trend of Pm none (t) for different values of Pi (t) as a function of the number of processors engaged.

Then, by increasing the number of processors it is possible to decrease the probability that solving an instance of the problem will take more than a fixed amount of time t. It is clear that even for poor (low) values of Pi(t), good (low) values of Pm~(t) can be reached as the number of processors increases.When we deal with issues like the random variable we must keep in mind that the probability to find out a feasible step for the physical model to actually move in physical state space is less than in reality.

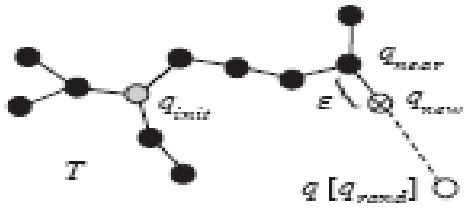


Fig. 2. Parallel splitting and joining of qnear and qrandom (or close) using dX/dt=f(X,u) control inputs.

This brings up the concept of latency and redundant computation which is unavoidable as it is linked to the core of the computation. A call to NEAREST_NEIGHBOUR (T, Xrand) finds out the path in linear time but its entire contribution to the algorithm is of order $O(K^2)$ where K is the tree size. Now, K may be large because of the redundant time spent in computing a feasible solution which enables the physical model to actually move keeping in mind that the movement has to take it closer in an optimal way toward the goal and not just wander anywhere which is the main issue we would be facing in a real practical situation. Hence the algorithm turns out to be as computationally expensive as $O([K_{red} + K_{feas}]^2)$ which increases to a huge value with $K \rightarrow \infty$. In practical situations this means a complete stall for the robot which waits there until a feasible path based on some biasing mode is found and this is practically unacceptable. Hence here we reduce the linear time complexity to a logarithmic order, although we cannot call it strictly logarithmic but the results achieved are good. We use a binary search tree to hold the edge distant metric so that ($\rho + \rho$ edge) can be quickly found out from the tree. At every step two trees are created in decreasing order of their (x, y) co-ordinates from the origin. We iterate back over the edges to find out which vertex from the minimum side coincides with its ordinal counterpart to give a feasible solution and hence the latency and probability factors come into the picture. By making it parallel (2(a) and 2(b)) the time required for the ith iteration is log (i/p) where p is the number of processors.Hence $O(\sum_{i=1}^{K}[\log(i/p) + \log(i/p) \times \log(i/p)])$ is

PARALLEL RRT GENERATOR(X, ρ , K) to be executed parallely and called in place of ADD

```
1    T.Init(Xinit);
2    INSERT (Xinit, Tsize, processor_id);
3    Xrand ← RANDOM_CONFIG ( );
4    PARALLEL NEAREST NEIGHBOUR (Tsize, Xrand);
5    u ← SELECT_INPUTS ( );
6    Xnew ← ∫f(x, u).dt+Xnear;
7    if Xnew feasible then
8      move;
9      add vertex (Xnew, T);
10     create communication, stop & reiterate with Xnew
11   else
12     wait ( );
13     if match with Xgoal then
14        exit ( );
15     else
16        continue;
17     End if
18   End if
19   RETURN T;
```

2 (a). PRRT GENERATOR Algorithm.

PARALLEL NEAREST NEIGHBOUR (Tsize, Xrand)

```
1    processor_id ← PROCESSOR_RANK ( );
2    INSERT (X, Tx, processor_id);
3    INSERT (X, Ty, processor_id);
4    while Tx → leftchild NOT NULL do
5         iterate till left ( xleft ) and ( xright)
6          Are found for xrand
7    End while
8    while Ty NOT NULL do
9         iterate till left ( xleft ) and ( xright)
10          Are found for xrand
11        while Tx → leftchild NOT NULL do
12          iterate till left ( xleft ) and ( xright) OR finds Ty
13          for which ρ is minimum from Xrand
14        End while
15   End while
16   If Ty NOT NULL then
17        RETURN Ty;
18   Else
19         RETURN NULL;
```

2 (b). PARALLEL NEAREST NEIGHBOUR Algorithm.

the net algorithmic complexity is for K>p otherwise also for communication delays arising due to non-blocking/blocking implementation-SMT issues etc. We use a binary search tree to hold the edge distant metric so that ($\rho + \rho$ edge) can be quickly found from the tree.

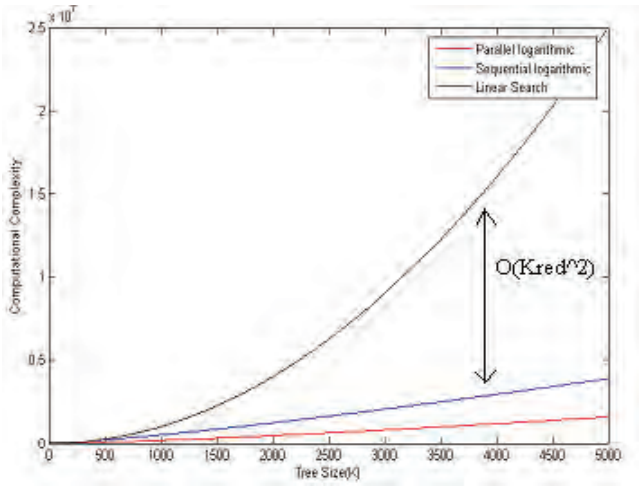


Fig. 3. Variation of computational complexity with data size (exaggerated at certain points to depict clarity). The redundant time being reduced effectively thus resulting sufficient decrease in real time stagnancy and latency in solution determination.

At every step two trees are created in decreasing order of their (x, y) co-ordinates from the origin. We iterate back over the edges to find out which vertex from the dominant term is different and is of a linear nature. A factor must be assumed to account for both latency of solution and the minimum side coincides with its ordinal counterpart to give a feasible solution.

4 Results and Simulations

4.1 Parallel Computer and Architecture

All the numerical results of the simulations carried out and illustrated in this section have been obtained by running the parallel algorithms on an HP D Class 9000, PA RISC Powered parallel computer based on *Dual Cluster* nodes. The computer is composed of 4 nodes connected by a high speed switch which warrants 500 Megabits of bandwidth. Every node includes 4 processors arranged in an SMP like architecture. Each dual cluster machine is connected by high speed LAN (bandwidth approximately 100 Mbps.) 2 Gigabytes of memory is shared in every node with clock speed 180MHz.Processors on different nodes can communicate and exchange data using the Message Passage Interface (MPI). The software has been developed in ANSI C++ and has been compiled using the highly optimized HP C++ compiler(S 800). This architecture is well suited for implementing the proposed parallel strategy previously illustrated. For the *process splitting* algorithm, each of the processors performs its computation independently and the termination message is sent or received using the MPI primitives. Every node executes the parallel algorithm according to the scheduled path (process splitting approach) and the first node which solves the problem sends a termination message to other nodes.

4.2 The Dynamical Vehicular Test model and Environment modes

The Dynamical Vehicular Test model used here for the simulation is a mechanical car with differential gearing and powered by a battery source. The model is connected to the computer with the quadruple mode dual bus that is responsible for controlling the car. Data input/output from the sensors placed on the car itself is transferred through the parallel I/O port using a Digital Stabilizing Driver circuit designed specifically for parallel ports. The sensors for detecting kinematics and dynamic constraints as well as obstacles in X are fitted at the front of the car. Here we have used an N x N grid of elements as the environment or X in which the motion of the model is considered. Here the black patches signify obstacles which are randomly placed and the model neither the controlling program/algorithm are aware of the exact locations.

Hence it is totally instantaneous and since our governing control equations is dX/dt=f(X , u) the vector {u} is supplied by the sensors which control the motion direction by first sending the input to the processors and then by feedback to the differential gearing and the circuitry to adjust the required changes which has been implemented on board. We have used infrared sensors which sense the change in infrared light intensity reflected from the obstacle.. Noise is reduced by using two operational amplifiers in series (LF353N-Dual) so that we can safely conclude that the car is able to detect the constraints effectively.

The figure below shows the model guiding its way through a single and also a multiple obstacle region. The lines and edges obtained below are an approximate one as we used numerical approximation to the control equation dX/dt=f(X, u) to obtain the edges. Figures 4(a) and 4(b) show some of robot motion and simulation results. Those preliminary results demonstrate the effectiveness of the proposed method. Figures 5 (a) ,(b) and (c) show the underlying principle of the motion sensors . The states are actually more smoothly connected than it appears because they are plotted only at every Δt.

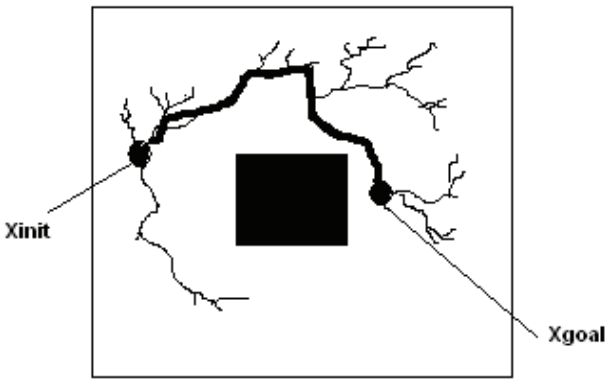


Fig. 4(a).

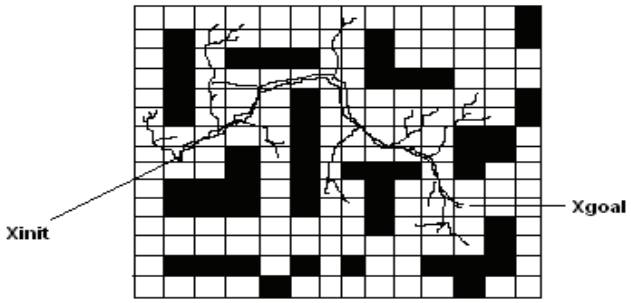
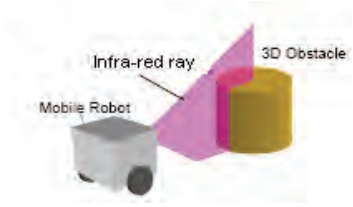
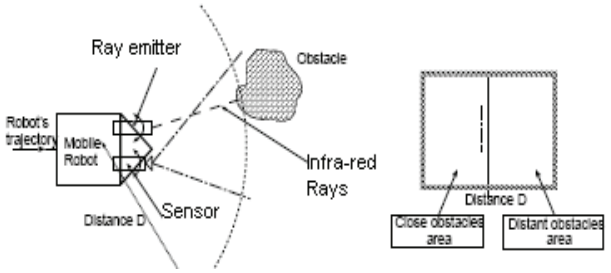


Fig 4(b) .

Fig 4.4(a) and (b) show the motion path of the model as it moves through the regions. The bold lines are the actual path leading from Xinit to Xgoal. The lighter ones indicate the generated tree vertices and the bold ones indicate the traversed path as plotted with an approximate data obtained from the simulation program.



(a) The emission of Infra-red rays.



(b) Top view with obstacle. (c) Sensor response.

Fig. 5. Simulation of the vehicular model moving around the single obstacle and obstacle detection principle.

4.3 The Motion Planning Problems Studied

The state of the robots are described as (x ,y, Θ) using its position x , y on the plane and orientation Θ. Given its goal state, the planner attempts to find a sequence of synchronization parameters of the modular robot to reach the goal.Although small numbers of values are used are used to reduce the search space, this simple combination turned out to be sufficient to generate various motions. Collision detection is implemented in the sensor handler code itself. According to each input, the next state (x , y, Θ) after time Δt = 2 (sec) are calculated by the dynamics simulator. We use relatively large Δt to allow the robot to make oscillatory motion for a certain period. Then the PRRT planner explores this state space to reach the goal. Here, goal-biased PRRT planner is used as the planner. In the initial state, the robot is at (0, 0, 0) and goal state is (20, 0, 0), namely moving to a given position with the same orientation, on in a plane that ranges from (−6, −10) to (24, 10). Two simulations are conducted where there are obstacles with different shapes and different numbers (single block and cluttered environment with multiple obstacles) at different positions.

4.4 Parallel Issues-Scalability, Speedup and Efficiency

An important aspect of performance analysis is the study of how algorithm performance varies with parameters such as problem size, processor count, and message startup cost or latency as in (Kumar, V. and Gupta, A., 1991), (Sun,X.-H. and Ni,L.M..,1990). In particular, we may evaluate the scalability of a parallel algorithm, that is, how effectively it can use an increased number of processors. One approach to quantifying scalability is to determine how execution time T and efficiency E vary with increasing processor count P for a fixed problem size and machine parameters. Execution time is not always the most convenient metric by which to evaluate parallel algorithm performance. As execution time tends to vary with problem size, execution times must be normalized when comparing algorithm performance at different problem sizes. Efficiency-the fraction of time that processors spend doing useful work---is a related metric that can sometimes provide a more convenient measure of parallel algorithm quality. It characterizes the effectiveness with which an algorithm uses the computational resources of a parallel computer in a way that is independent of problem size. We define *relative efficiency* as

$$\mathbf{E= (Ts/Tp)/p} \qquad (3)$$

where Ts is the sequential time i.e. time to run the code on a single processor system and Tp being the time taken to run over p processors parallely. The *relative speedup* is defined as

$$\mathbf{S=Ts/Tp} \qquad (4)$$

A common observation regarding parallel processing is that every algorithm has a sequential component that eventually limits the speedup that can be achieved on a parallel computer. This observation is often codified as *Amdahl's law*, (Amdahl, G.,) which can be stated as follows: if the sequential component of an algorithm accounts for *1/s* of the program's execution time, then the maximum possible speedup that can be achieved on a parallel computer is *s*. The following *f*igures show the comparison of speedups as also net speedup gained over the *initial linear search* algorithm with the *parallel* algorithm.

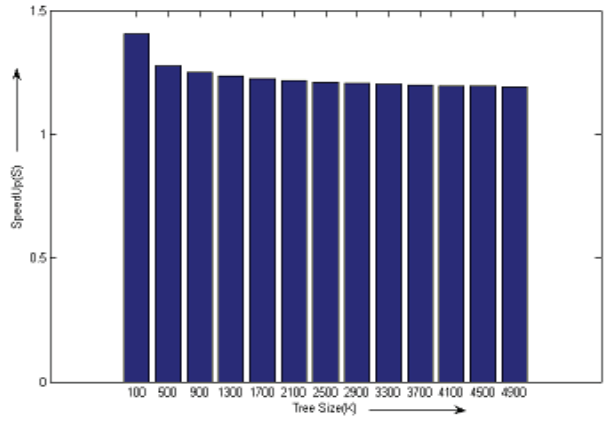


Fig 6(a).Speedup for p=2.

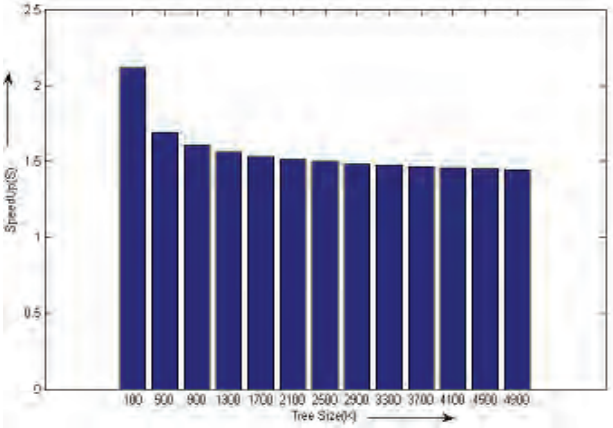


Fig 6(b). Speedup for p=4.

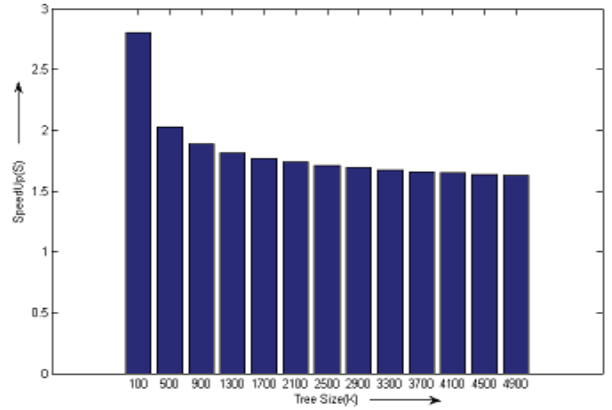


Fig 6(c) .Speedup for p=6.

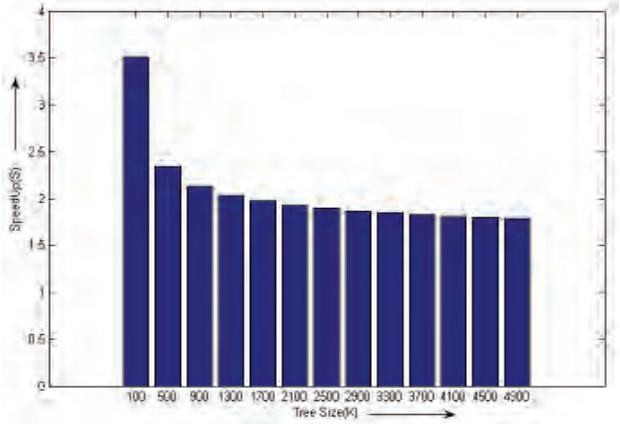


Fig 6(d).Speedup for p=8.

Fig 6. (a),(b) ,(c) and (d) show performance comparisons of the speedup in comparison to the logarithmic algorithm achieved for varying problem size(RRT-length) with p constant and for p=2,4,6 and 8.The maximum achieved speedup being close to 3.5 for p=8.All tree sizes are considered by running the test a number of times and selecting only the times that correspond to the vehicle reaching closest to Xgoal for a particular approximated K.

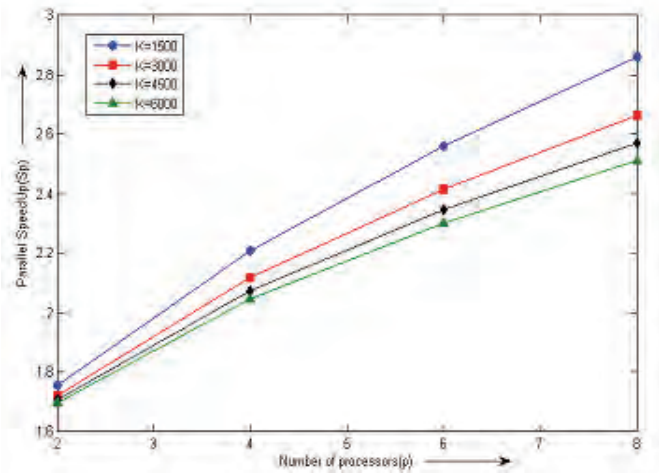


Fig 7(a).

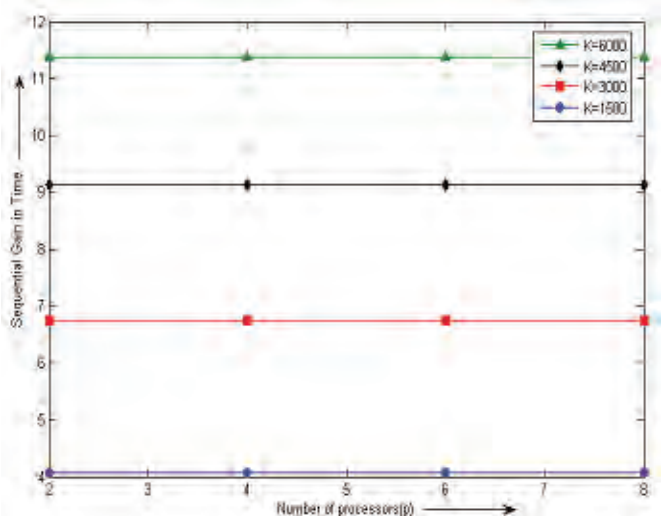


Fig 7(b).

Fig 7. (a) and (b) show the speedup achieved in comparison to the logarithmic as well as the linear algorithm with increasing number of processors ,p=2,4,6 to 8 on the HP S 800 compiler with Dual Cluster for various tree sizes.

We also consider the error encountered in calculating the feasible angle of turn Θ between the one calculated from the program itself by assuming a known function Ψ(X) which gives the obstacle information as true/false and the experimental case where it is detected by the infrared sensors on the robot in real time.

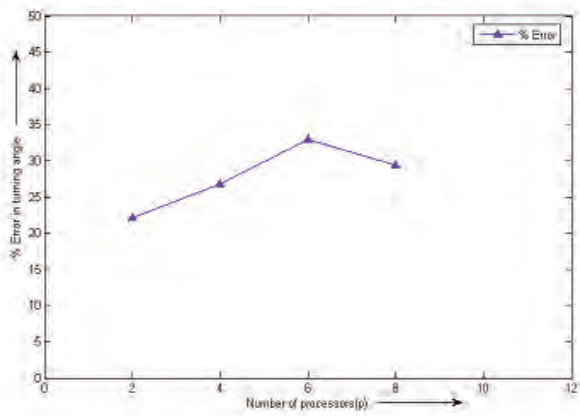


Fig 8.Figure shows the plot of %error encountered in the feasible angle of turn which is biased towards the goal between Θ theo and Θ exp.

5. Conclusions

In this paper we illustrated how it is possible to improve the performance of RRT based motion planners using a parallel approach. Numerous techniques can be applied, namely classical parallel and cooperative embarrassingly parallel computation. More efficient algorithmic techniques are needed to develop the NEAREST_NEIGHBOUR method. Much efficiency in computational terms can be achieved by creating multiple PRRTs (for example, one rooted at x init and another rooted at x goal).A PRRT could replace the random walk stage in a randomized potential field approach. For some problems, it might be preferable to obtain multiple, homo-topically distinct paths. In this case a PRRT could be converted into a cyclic graph and also to reduce the inevitable *Real Time Stagnancy* and *parallel latency* issues.

At the moment we can only claim to have achieved the parallelization to some extent; but its true power and form is unfathomable, which if implemented much more efficiently can lead to a lot better and computationally efficient autonomous system with innate navigating capabilities. Assembly –level parallelization can bring a break-through in autonomous robot development and effective embedded planners with much less cost and labor required to achieve the complexities involved in developing a parallel system at this level of complication.

6. References:

Arkin, R. C. Motor schema-based mobile robot navigation.International Journal of Robotics Research, August 1989, 8(4):92–112, 1989.

Atramentov, A. and LaValle, S. M. Efficient nearest neighbour searching for motion planning. Submitted to 2002 IEEE International Conference on Robotics and Automation, 2002.

Koenig, S. Agent-centered search. Artificial Intelligence, 2002, in print.

Latombe, J.C. Robot Motion Planning. Kluwer, 1991.

LaValle, S. M. Rapidly-exploring random trees: A new tool for path planning. In Technical Report No. 98-11, October, 1998.

LaValle, S. M. and Kuffner, J. Randomized kinodynamic planning. In International Journal of Robotics Research, Vol. 20, No. 5, pages 378–400, May 2001.

Stentz, A. Optimal and efficient path planning for unknown and dynamic environments. In International Journal of Robotics and Automation, Vol. 10, No. 3, 1995.

Donald, B., Xavier, P., Canny, J., and Reif, J. Kino dynamic motion planning. *Journal of the ACM*, 40(5):1048–1066, November 1993.

Latombe,J.C.. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *The International Journal of Robotics Research - Special Issue on Robotics at the Millennium* , 18(11):1119–1128, 1999.

Reif,J.H.. Complexity of the mover's problem and generalization. In *Proceedings of the 20th IEEE Symposium on foundations of Computer Science*, pages 421–427, 1979.

Kavraki, L.E., ˇSvestka, P., Latombe, J.C., and Overmars, M.H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1998.

Henrich, D. A review of parallel processing approaches to motion planning. In *Proc. of ICRA*, pages 3289–3294, 1996.

Kuffner,J., Nishiwaki, K., Kagami, S.,Inaba, M. and Inoue,H. Motion planning for humanoid robots under obstacle and dynamic balance constraints. In *Proc. of ICRA*, pages 692–698, 2001.

Caselli, S. and Reggiani, M. Erpp: An experience-based randomized path planner. In *Proc. of ICRA*, pages 1002–1008, 2000.

Challou,D., Boley,D. , Gini,M.,Kumar, V., and Olson,C. Parallel search algorithms for robot motion planning. In K. Gupta and A.P. del Pobil, editors, *Practical Motion Planning*, pages 115–132. JohnWiley & Sons,1998.

Kumar, V. and Gupta, A, "Analysis of scalability of parallel algorithms and architectures: A survey," in Proc. Int. Conf. on Supercomputing, June 1991.

Sun, X.H. and Ni, L.M., "Another View of Parallel Speedup," Proc. Supercomputing '90, IEEE Computer Soc. Press, Los Alamitos, Calif., 1990, pp. 324-333.

Amdahl, G., "Validity of the single-processor approach to achieving large scale computing capabilities," in Proc. AFIPS Conf., 1967, pp. 483-485.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E. Robocup: The robot world cup initiative.In Proceedings of the IJCAI-95 Workshop on Entertainment and AI/ALife, 1995.