Künstl Intell (2016) 30:245–255
DOI 10.1007/s13218-016-0450-z

TECHNICAL CONTRIBUTION

Full-Body Motion Planning for Humanoid Robots using Rapidly Exploring Random Trees

Jacky Baltes[1] · **Jonathan Bagot**[2] · **Soroush Sadeghnejad**[3] · **John Anderson**[2] · **Chen-Hsien Hsu**[1]

Received: 9 May 2016 / Accepted: 23 July 2016 / Published online: 1 August 2016

Abstract Humanoid robots with many degrees of freedom have an enormous range of possible motions. To be able to move in complex environments and dexterously manipulate objects, humanoid robots must be capable of creating and executing complex sequences of motions to accomplish their tasks. For soccer playing robots (e.g., the participants of RoboCup), the highly dynamic environment require real-time motion planning in spite of the enormous search space of possible motions. In this research, we propose a practical solution to the general movers problem in the context of motion planning for robots. The proposed robot motion planner uses a sample-based tree planner combined with an incremental simulator that models not only collisions, but also the dynamics of the motion. Thus it can ensure that the robot will be dynamically stable while executing the motion. The effectiveness of the robot motion planner is demonstrated both in simulation and on a real robot, using a variation of the Rapidly Exploring Random Tree (RRT) type of motion planner. The results of our empirical evaluation show that CONNECT works better than EXTEND versions of the RRT algorithms in simple domains, but that this advantage disappears in more obstacle-filled environments. The evaluation also shows that our motion planning system is able to find and execute complex motion plans for a small humanoid robot.

✉ Jacky Baltes
jacky.baltes@ntnu.edu.tw; jacky.baltes@gmail.com

[1] National Taiwan Normal University, Taipei, Taiwan, ROC

[2] University of Manitoba, Winnipeg, MB, Canada

[3] Amirkabir University of Technology, Tehran, Iran

Keywords Humanoid robots · Motion planning · RRT · Real time AI

1 Introduction

The cost of building robots has decreased considerably in recent years. Now many researchers are able to develop humanoid robots, and address new research challenges with robots possessing increased degrees of freedom (DOF).

A desire to increase the robot's range of motion and autonomous performance in uncertain environments with real-time requirements makes motion planning a crucial component of an intelligent robot system. In robotic soccer competitions such as RoboCup, a small set of parameterised motion plans are currently sufficient (e.g., walk forward, walk backward, turn left, turn right, shuffle left, shuffle right, kick left, kick right, stand up forward, and stand up backward). These can be implemented off-line in about two weeks by a motivated graduate student. However, in the future, improved skills of the robots will require more versatile motions (e.g., headers, squeezing by an opponent, throw-ins, and sliding tackles). As the robot begins to be capable of performing a large number of complex motions, it becomes impossible to pre-program all possible motions offline. Instead the robot must generate new motions online. Efficient motion planning is also necessary for greater utility and to allow robots to work in shared work spaces.

In this paper we use Rapidly Exploring Random Trees (RRTs) as a basis for complex online motion planning, implementing several variants and examining these in both simulation and on a 19 DOF physical humanoid robot in five different environments.

2 Background and Related Work

The motion planning problem is a form of the general mover's problem, an extension of the piano mover's problem. The piano mover's problem is to determine how to move a piano from an initial position to a goal position without colliding with objects in the environment. The general mover's problem replaces the piano with a generic object where the moving object may have multiple polyhedra freely linked together at various distinguished vertices [14].

Planners generally rely on some form of representation to reason about space in the world and the consequences of their actions. The most commonly used state space representation for robot motion planning is known as a Configuration Space (CSPACE) [13]. A robot in CSPACE is reduced to a single n-dimensional point (configuration) moving through space, where n is the number of DOF. The CSPACE represents the set of all transformations that can be applied to the robot. The Configuration Free (CFREE) set denotes the set of all robot configurations that are not in collision with obstacles. The Configuration Obstacle (COBS) set is the compliment of the CFREE set, therefore the COBS set denotes the set of all robot configurations that are in collision with obstacles. A valid path from a robot's initial configuration to goal configuration would consist of configurations which all belong to the CFREE set.

Solving a planning problem for a full-bodied articulated humanoid robot is difficult, because the search space yields a combinatorial explosion. Consider a subset of the general humanoid motion problem, an assembly robot arm such the Selective Compliance Articulated Robot Arm (SCARA) with 4 DOF, as a simple example. If each of the 4 joints has a range of [0.0, 180.0] in 0.1 degree increments, this yields a state space of 1800^4 for just this arm. Finding the optimal solution would require searching through the state space for a set of states that arrive at the goal state from the initial state and maximize a heuristic function. The heuristic function itself can also have many factors: for example if a robot has a limited power supply, the heuristic function might rank solutions that use less power more highly. Searching through the entire state space is infeasible for high-DOF humanoid robots especially if a solution must be found in real-time, and so for robots with many DOF in uncertain environments, completeness and optimality are typically sacrificed [6, 8].

By giving up completeness, we accept that the path planner will sometimes not find a path, even if such a path exists. Reducing a robot's think and react cycle time is a critical task for operation in dynamic environments, and a path that is good enough but not optimal supports this. Sample-based probabilistically complete planners (i.e., will find a solution if one exists but cannot determine if one does not exist) have shown promise returning sub-optimal results in reasonable time. Sample-based planners can be classified in two general groups: roadmap and tree [18]. The main difference between roadmap and tree planners is the underlying data structure. A roadmap planner typically stores the map in a graph and requires two phases to return a solution [3]. The first (learning) phase builds the map and the second (query) phase determines the plan with a graph search algorithm. The learning phase is an expensive operation but in a static environment the learning phase needs to run only once. In dynamic environments the cost of the learning phase would be too large to react to changes in the environment in a timely fashion, since the roadmap must be rebuilt before each query [4]. A tree planner stores only the necessary data to find a solution to the query in a tree. The root of the tree represents the initial state and the tree is built incrementally until the goal state is reached. The answer to the query is simply a path from the root (initial) to a leaf (goal) node [17]. Building the tree is computationally inexpensive which makes tree planners ideal for dynamic environments where plans may become invalid quickly.

A popular version of sample-based roadmap planning is the Probabilistic Road Map (PRM) [2, 7]. The learning phase creates the PRM by randomly generating configurations in CFREE which are then connected to nearby configurations. The learning phase does not consider the initial or goal configuration. A map for the entire CSPACE is created, which is a waste of time if the environment changes. For this reason, roadmap planners work best for holonomic robots in static environments where the map can be queried multiple times once generated because of the cost incurred by the learning phase [10, 11]. The PRM is a multi-query planner, whereas the solution strategy presented in our work is a single-query planner that does not require a learning phase. Our solution strategy only considers the portions of the environment necessary to find a suitable motion plan.

A major disadvantage of using PRMs is the number of connections that must be made between configurations, which is not a simple task [10]. Our solution strategy does not require as many connections between configurations, which reduces the time to find a suitable motion plan. For a large CSPACE the learning phase is an expensive operation because of the logic required to make connections. In an uncertain environment with real-time requirements, the learning phase (think) does not allow sufficient time to react and the map cannot be reused if the environment changes rapidly.

Many popular versions of sample-based tree planners use the Rapidly Exploring Random Tree (RRT). An RRT is

a data structure where the nodes of the tree represent configurations. The configurations are generated randomly and typically biased towards unexplored areas [10]. The RRT is rooted at the initial configuration of the robot. In a tree, every node except the root has exactly one parent node. The random configurations in a RRT are connected in this fashion. Each random configuration or node in the RRT belongs to the CFREE set. Every edge between nodes in a RRT represents a path which also belongs to the CFREE set. The goal is to generate a leaf node which is the goal configuration. The RRT data structure itself can be used as an algorithm for motion planning because the data structure translates directly to the CSPACE state space representation. A path in the tree from the root (initial configuration) to leaf (goal configuration) is a motion plan. The RRT algorithm terminates once the goal configuration is achieved, so no unnecessary work is done. A RRT does not require a learning phase like the PRM. Only a single phase or query is required to determine a motion plan, and this more easily supports dynamic environments.

The key parameters of a RRT are a sample bias method, a method for selecting the nearest neighbor given a random node (NN method), a distance metric, and a method for collision detection. The choices for each of these elements affect the performance of the RRT in different ways. For example, different sample biases have a direct impact on how well the tree covers the environment while the nearest neighbor method only influences how fast random configurations can be connected to the tree. In this work we use RRTs as a basis for a complex motion planner, and contrast a number of variations in RRT approaches.

In the classic RRT Algorithm [10], the root of the tree is set to the robot's initial configuration (pose and coordinates). The algorithm then adds random configurations to the tree n times, taking care to prevent premature termination due to small n. These random configurations are generated according to some sample bias, and the nature of a configuration is dependent on the robot—if a robot has 19 DOF, for example, each configuration will consists of the randomly generated joint angles for all 19 DOF. Once the random configuration is generated, the nearist neighbour (NN) in current tree is found using a specified NN and distance metric method. After the NN is found, a move is attempted from the NN to the random configuration. One of three things may occur during the attempted move between the two configurations. Either the move is blocked by an obstacle, which is determined by using a specified collision detection method, or the move is not physically possible with the robot, or the move can be performed by the robot. If the move can be performed by the robot then the random configuration is added to the tree by connecting it to the NN. If the move is not possible then a new configuration that was generated by the movement up until it could no longer get any closer to the random configuration is added to the tree by connecting it to the NN.

Variants of the RRT data structure as a motion planner, such as RRT-CONNECT have been shown to be probabilistically complete [11]. However is has been proven that the probability of the classic RRT construction algorithm converging to an optimal solution is zero [5]. When a problem has a solution, a probabilistically complete algorithm's probability of finding a solution goes to one as the runtime approaches infinity [1]. In other words, the algorithm will eventually converge to a solution. In our work, the goal is not necessarily to find the optimal solution in terms of a heuristic function. The main concern is finding a sub-optimal solution that is good enough as quickly as possible. Further variants of the RRT are possible, such as RRT-EXTEND, where the move towards the random configuration from the NN is performed for n time steps, or until the NN is reached, or an obstacle is blocking the path.

Motion planning algorithms also differ in producing results that are stable in the environment, beyond simply a collision free path from an initial state to a goal state. Stability is relatively straightforward for simple wheeled robots where the terrain is even. This becomes difficult very quickly once terrain becomes uneven, and where robots themselves are not dynamically stable in all possible configurations (e.g. humanoids with many DOF operating in real time). Our motion planner includes kinodynamic constraints, can find or approximate the inverse kinematics (IK) solution, and performs collision detection quickly and reliably in order to generate feasible motions plans for real time processing.

3 Motion Planner with RRT and Incremental Simulator

Because the intended use of our motion planner is for high-DOF humanoids, we begin by describing the humanoid robot for which it was developed, for context. The robot *Blitz* used in this research is shown in Fig. 1. Blitz is a custom modified robot based on Robotis' Bioloid kit, with 19 DOF. We used a Nokia 5500 mobile phone running Symbian 9.1 to control the robot. This has a reasonably powerful CPU that executed the motion planner and performs visual processing with the phone's camera used for visual capture. The Nokia 5500 communicates with a

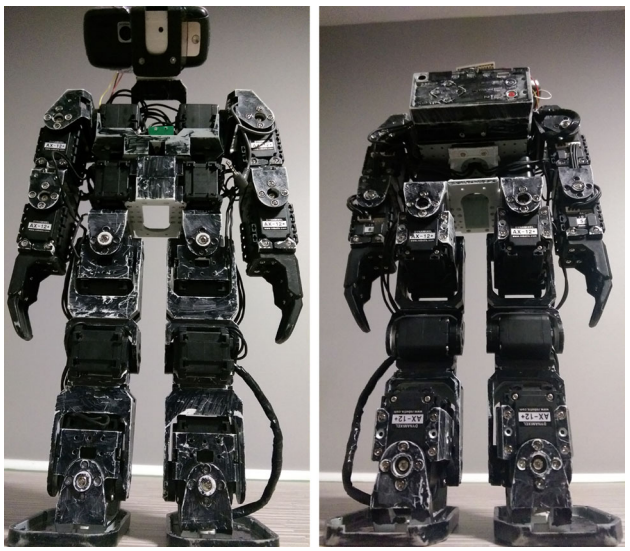

Fig. 1 Humanoid robot blitz

custom-built IRDA transceiver with an AtMega128-based microcontroller that is responsible for sending commands to the actuators. A three-axis gyroscope and accelerometer in the Nokia 5500 is used for balancing.

Each DOF in blitz is articulated by a Dynamixel AX-12 servo, which is capable of producing 1.5 N-M of torque at 12V. The servos are daisy chained together using a serial bus.

Our motion planner is a single-query algorithm, since it does not require building a map before a solution can be found, unlike other approaches such as Probabilistic Road Maps (PRM) [7]. The solution is found by incrementally building a data structure, which ultimately encompasses the solution.

The single-query sample-based tree algorithm employed is a variant of the RRT construction algorithm, RRT-EXTEND and RRT-CONNECT [9, 11]. Pseudocode for these are shown in Algorithms 1 and 2. The RRT-EXTEND algorithm differs from the classic RRT in that the move towards the random configuration from the NN is performed for n time steps, or until the NN is reached, or an obstacle is blocking the path. The RRT-CONNECT algorithm is similar to the EXTEND algorithm but there is no restriction of n time steps. Without this restriction, the CONNECT algorithm can generate longer paths with fewer calls to the NN method. CONNECT tends to work best for holonomic planning problems, while EXTEND tends to work best for non-holonomic planning problems.

Algorithm 1 RRT-EXTEND 1 [9,11]

procedure BUILD($init_c, n, \delta t$)
  $init_c \leftarrow RRT.init()$ ▷ Robot initial configuration.
  for $i \leftarrow 0, n$ **do**
    $rand_c \leftarrow RandomConfiguration()$ ▷ Uses sample bias method.
    $EXTEND(RRT, rand_c)$
  end for
  return RRT
end procedure
procedure EXTEND($RRT, rand_c$)
  $near_c \leftarrow NearestNeighbor(rand_c, RRT)$ ▷ Uses kd-tree.
  $motion \leftarrow Move(rand_c, near_c)$ ▷ Minimum path motion.
  $new_c \leftarrow NewConfiguration(near_c, motion, \delta t)$ ▷ Motion from $near_c$.
  if $new_c \neq near_c$ **then**
    $RRT.addConfiguration(new_c)$ ▷ Adds configuration to the tree.
    $RRT.addEdge(near_c, new_c, motion)$ ▷ Path between two configurations.
    if $new_c = rand_c$ **then**
      return $REACHED$ ▷ Successful connection.
    else
      return $ADVANCED$ ▷ Partially successful connection.
    end if
  else
    return $TRAPPED$ ▷ Unsuccessful connection.
  end if
  return RRT
end procedure

Algorithm 2 RRT-CONNECT [9,11]

```
1: procedure BUILD(init_c, n, δt)
2:     init_c ← RRT.init()                          ▷ Robot initial configuration.
3:     for i ← 0, n do
4:         rand_c ← RandomConfiguration()           ▷ Uses sample bias method.
5:         CONNECT(RRT, rand_c)
6:     end for
7:     return RRT
8: end procedure
9: procedure CONNECT(RRT, rand_c)
10:    repeat
11:        result ← EXTEND(RRT, rand_c)
12:    until result ≠ ADVANCED
13:    return result
14: end procedure
```

Our motion planner builds an RRT while exploring the CSPACE, and nodes in this tree represent robot configurations and the current state of our incremental simualtor. This results in a vast search space which makes exhaustive or even systematic search intractable. However, the size of the search space alone is not the only important feature that determines the runtime of the motion planner. Another important feature is the number of possible solutions in the search space. In puzzles, there are few solutions and common sense heuristics often lead the search astray, but in motion planning, there are usually many possible solutions and reasonable heuristics such as the distance to the goal. So in this research, we show that a probabilistic algorithms with fast checks for validity provided by an inremental simulator are able to find good solutions.

Robot configurations are randomly generated and the planner attempts to move from the closest configuration currently in the tree to the new configuration.

If the resulting configuration is legal (that is obeys all kinematic, dynamic, and stability constraints checked by the simulator), then the resulting configuration is added to the tree. The configurations that are legal and hence subsequently added to the RRT are those that: belong to the set of all robot configurations that do not result in collisions with obstacles (CFREE), and the robot configuration is dynamically stable, and the robot configuration is feasible given the torque limits of the robot.

Just as in any standard RRT, our planner requires a sample bias method, a NN method (the method used to find the nearest node to a randomly selected node), a distance metric, and a collision detection method. The standard implementation of a RRT has an inherent Voronoi bias when using uniform sampling because the probability that a configuration in the tree is selected is proportional to the volume of its Voronoi region [12]. Configurations with larger Voronoi regions are more likely to be chosen. The convergence of the standard RRT implementation can be improved with a goal bias [11] which simply selects the goal configuration instead of a random configuration using a predefined probability. The predefined probability must be chosen with care because a probability that is too large can potentially lead to local minima traps. The sample bias method used is a goal bias method. If the goal bias probability is set to zero then the RRT has the standard Voronoi bias. The NN method uses a third party library that can be configured for k-NN and different distance metrics. The collision detection method is built into the physics engine, and uses a mesh method similar to the bounding polygon collision detection method.

Our motion planning system also allows the sample bias method, distance metric, k-Nearest parameter, collision detection method, and incremental time step to all be dynamically altered during the planner's execution.

One novel feature of our motion planning system is the tight integration of a fast, accurate incremental simulator. The simulator checks kinematic constraints (e.g., collision detection and actuator range limits), but more significantly it checks dynamic constraints (e.g., torque limits on actuators). The simulator also implements the Inverse Kinematics (IK) of the robot. That is it determines the joint angles of the robot in order to achieve a desired target position and orientation. There is no general closed form solution to the IK problem for robot manipulators with greater than 6 DOF [16], therefore numerical solutions or hill climbing methods are used in such situations [15]. Since the simulator is an incremental simulator, a solution is found by incremental hill climbing. A nice side effect of this approach is that the increments are actually a coarse trajectory plan. Finally, the simulator calculates the zero moment point ZMP and uses it as stability criterion to ensure that the robot will not fall over while executing the motion.

The main feature of the simulator is that the simulation can be quickly restarted from a given snapshot in time to simulate the effects of a particular motion. The simulator creates snapshots that include the current kinematics (i.e., position of all joints) and dynamics (i.e., forces on the

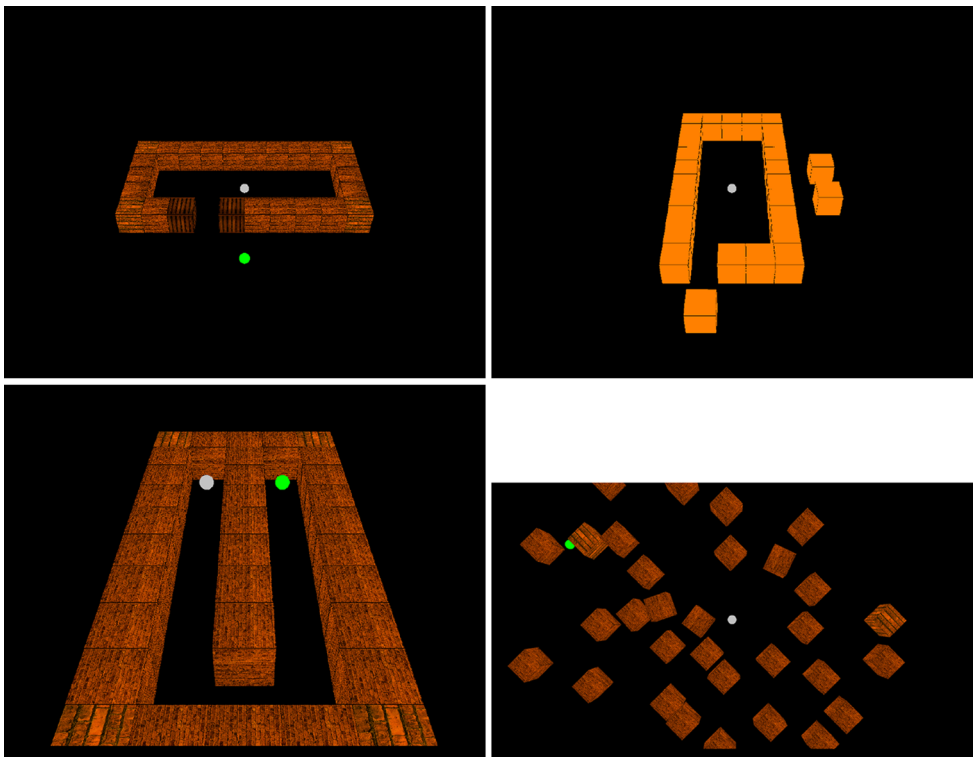

Fig. 2 Sample worlds used in the simulated portion of the empirical evaluation: World1 (*top left*), World2 (*top right*), World3 (*bottom left*) and World4 (*bottom right*)

joints). Only states that guarantee the stability of the robot are considered for inclusion.

The states in the RRT include the snapshot of the kinematic and dynamics of the simulator from the initial configuration to the node.

In the empirical evaluation that follows, we evaluate both both EXTEND and CONNECT-based versions of our motion planner.

4 Empirical Evaluation

We performed our empirical evaluation using both our simulator and the real world. Our evaluation was focused on two goals. The first was to showcase the versatility of the motion planner and simulator integration and to analyze some of the trade-offs associated with different parameters (sample bias, nearest neighbor collision detection method, distance metric, and connect heuristics). The second goal was to identify bottle-necks in the implementation and to find the fastest implementation methods so that the resulting motion planner is able to execute on the Bioloid robot platform in real-time.

We used two robots in this evaluation. In simulation only, we employed Sphere, a simple proof of concept spherical robot with six DOF—the translational and rotational velocities in the X, Y, and Z planes respectively. In both simulation and the real world we employed blitz, the humanoid robot described in Sect. 3.

We created six different simulation environments to evaluate the performance of the sphere and blitz robot of our motion planning system.

4.1 World Models

A world model consists of simple objects such as cubes, cylinders, planes, and spheres combined together to create a complex environment for a robot to traverse and manipulate, along with an initial configuration and goal configuration in terms of the arrangement of these objects and the robot's position and pose. The forces that act upon these objects, such as friction and gravity, are simulated using the physics engine in the motion simulator. The motion simulator user interface provides an easy method to create, modify, and arrange objects in the world. Four of the worlds used in our evaluation are shown in Fig. 2. World1 has the robot located inside of a box and the goal is just outside the only exit. World2 is similar, but has the goal moved to the side of the box and extra obstacles surrounding the goal position added. World3 is a u-shaped long and narrow corridor. World4 has a random field of boxes between the initial and the goal position. In our experiments, there is never a direct collision-free path from the initial to the goal configuration.

Real world environments that were used in the validation of our motion planning system with our humanoid robot blitz are shown in Fig. 3. The goal sits on top of the box, and must be touched by the tool manipulated by blitz.

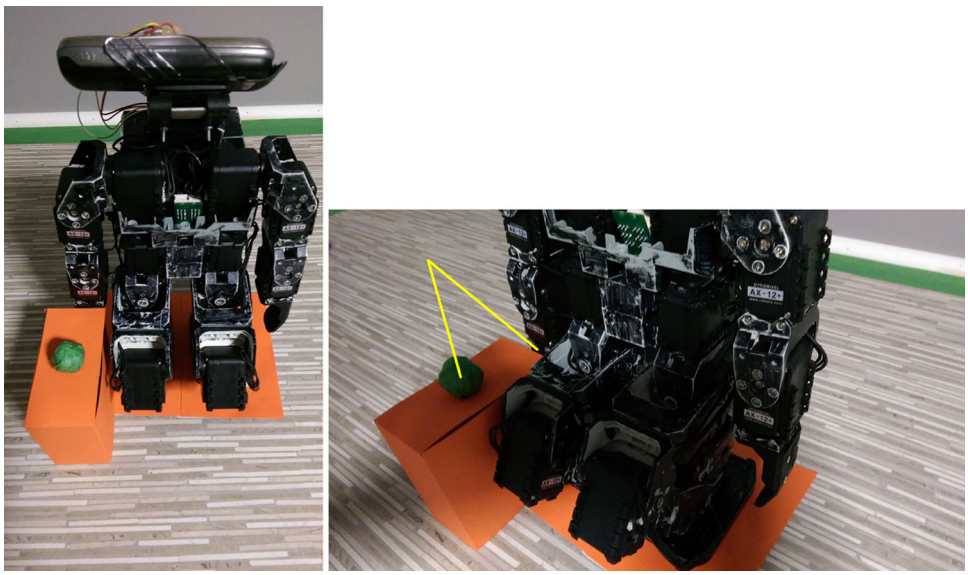

Fig. 3 Real World Environments used in the evaluation of blitz

There is no straight path directly from the initial configuration to goal configuration.

4.2 Experiments

One of the design goals of our motion planning system is to support real-time applications. We therefore investigated the run-time performance of many different goal biases, NN methods, and distance metrics, both in simulation and the real world, and found that a Euclidean distance metric with a 1-NN selection and a variable goal bias with 10 % increments resulted in the best real-time performance. Due to space limitations, we omit a detailed description of those experiments and focus on the trade-offs between EXTEND and CONNECT variants of the RRT algorithms.

Each combination of settings was performed three times in simulation. For the real robot each combination of settings was performed once due to the amount of time it takes to execute the plan on the real robot. We evaluated the performance of the motion planning system using the dimensions indicated in Table 1.

For the experiments the visualization of the motion plan provided by the motion simulator was used to view the plan as it was generated in real-time. The visualization is implemented in OpenGL and consumes significant resources when drawing many objects. It is expected that if the visualization was turned off that an improvement in total execution time can be realized.

While the plans produced for the Bioloid robots were large due to the number of motion changes, those for the Sphere robot can be reproduced reasonably here to show the plan complexity for this environment. Figures 4 and 5 show the best solutions found by the CONNECT and EXTEND algorithms respectively in the World0 environment for the sphere robot. In both cases, the fastest solution was found with a goal bias of 40 (but both algorithms returned reasonable paths within milliseconds using a goal bias between 10 and 40 percent). The figures do not show the contents of each node (i.e. the motion changes) but do show the path and and the underlying tree data structure (yellow nodes are on the correct path, red are added to the RRT but do not ultimately form part of the path) for the

Table 1 Summary of evaluation criteria

Criteria	Description
Total execution time	The motion planner's response time to a plan request
Number of configurations	The number of configurations in the motion planner's data structure
Modified random configurations	The number of generated random configurations that had to be modified before use in the motion planner's data structure
Dispersion	A measure of how distributed the configurations are throughout the world [12]
Total move test time	The total amount of time to check if a move between to configurations is possible
Total NN query time	The total amount of time to query the NN data structure
Qualitative inspection	Quality of the generated plan by manual inspection of the output

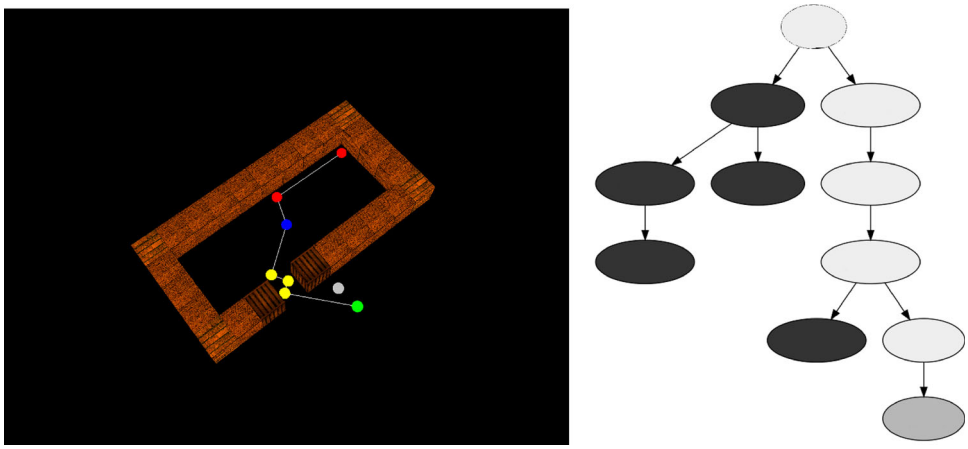

Fig. 4 Path and search tree generated by CONNECT (Goal Bias 40) in World0

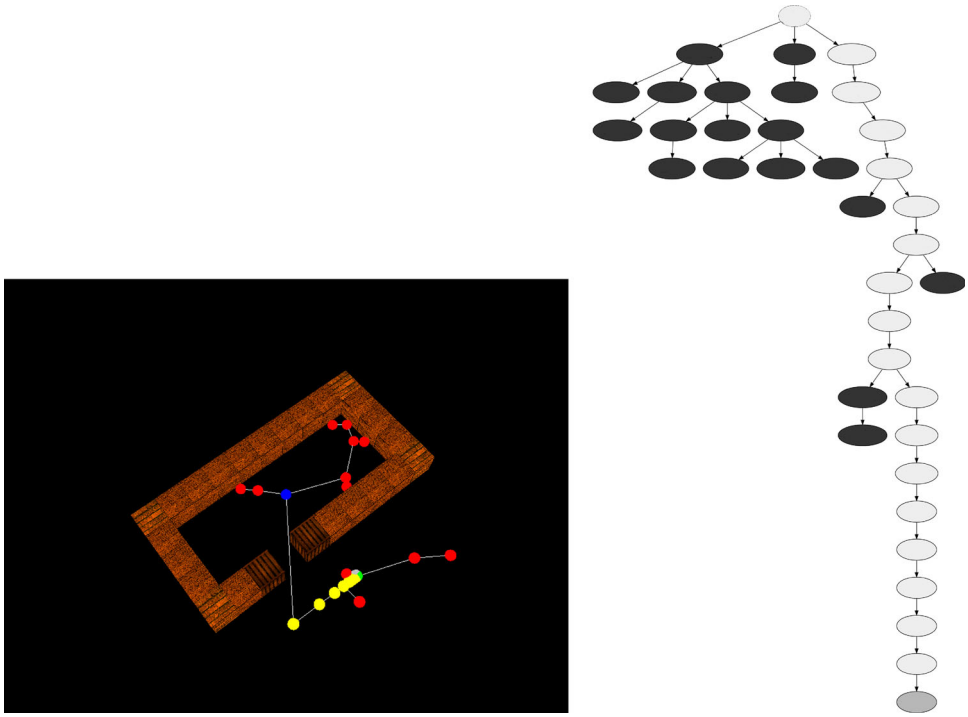

Fig. 5 Path and search tree generated by EXTEND (Goal Bias 40) in World0

motion planner. The CONNECT algorithm out performed the EXTEND algorithm for this relatively simply case. Since the goal configuration is not very far, adding random configurations to the RRT that are far away from the goal configuration actually causes a loss of performance for the CONNECT algorithm. The EXTEND algorithm does not add random configurations that are very far since motion from NN is only attempted for n time steps before modifying the random configuration so this is actually advantageous in this case. We found similar results for environments World1 and World2.

The total execution time of World0 was dominated by the total move test time, accounting for 97 % of it. This was a common finding for all experiments. Due to the implementation of the move test (IK and trajectory planning solutions), determining if the robot can move between two configurations is slow since the incremental simulator is stepped in real-time. This could easily be solved by stepping the incremental simulator faster than real-time, or if the move test could be done with the models alone by using parametric equations. These improvements were not be considered at this time in our work since IK and trajectory planning solutions are not the focus of this research. However, if we look at the delta between the total execution time and total move test time, then the fastest solution average returned by the motion planner for a specific combination of settings was given in 0.2 s.

Based on the NN Query Time statistic for World0, the NN method is efficient and does not significantly impact the runtime of the motion planner. The majority of NN

queries returned in sub-millisecond time. The NN method is actually so efficient that there is no need to optimize its performance since any gains would be negligible. This is a common finding across all experiments. From these findings it was decided not to tweak NN epsilon which allows for the NN query algorithm to stop when close enough instead of calculating the exact NN. It was also decided to not change the Euclidean distance metric which is used exclusively by the NN method. If the robots used for the experiments had substantially more DOF, then the NN method might have a noticeable impact on the runtime of the motion planner. It is expected that the number of DOF necessary for there to be a noticeable impact on the runtime of the motion planner is an unrealistic amount for a real robot. Determining the threshold for the NN method and the number of DOF is not the focus of this research since it will not be applicable to real robots.

The majority of the random configurations in World0 are modified because of the way the EXTEND algorithm works. The number of modified random configurations provides a good indication of how cluttered with obstacles the world is, if most of the generated random configurations are in COBS as opposed to CFREE. The number of modified random configurations can also provide insight into how good the motion planner is at selecting configurations: if it is very good then no random configurations would need to be modified and they could be used directly. The motion towards the random configuration from the NN is only performed for n time steps, unlike CONNECT which will complete the motion unless an obstacle is hit or the robot cannot physically perform the motion. The number of modified configurations for EXTEND could potentially be reduced by increasing n. This is also a common finding across all result tables. As n is increased, the EXTEND algorithm will start to behave more like the CONNECT algorithm.

The tree images clearly illustrate one of the major difference between the EXTEND and CONNECT algorithms. EXTEND generates longer branches than CONNECT. For configurations that are far apart, EXTEND must generate many intermediate configurations between the two configurations where as if it is possible to move between the two configurations CONNECT does not require any intermediate configurations. Having the intermediate configurations could be useful if a trajectory planner could use the additional information to refine the trajectory, otherwise due to the wasted time generating intermediate configurations makes CONNECT a better choice than EXTEND.

When the dispersion is low and the number of configurations is also low, then typically the total execution time is small. In this case a solution was found quickly. A low dispersion is not necessarily bad since the goal of the motion planner is to find a solution rather than to do a good job at exploration. If dispersion is low and the number of configurations is high then this would definitely be bad because this would mean that the motion planner is trapped and is not doing a sufficient job of exploring to get out of the local minima. The latter was not observed in any of the experiments conducted.

It is desirable for the motion planner to return results in approximately the same amount of time when the same settings are used, while a large variability with the same settings is undesirable. This may not be completely unavoidable since a motion planner is probabilistic and will converge to a solution eventually if one exists, but a potential workaround if the motion planner is taking longer than expected to find a solution is to start the motion planner again with a different Random Number Generator (RNG) seed value because there is a chance that it will converge more quickly.

The motion planner had the greatest difficulty with the Sphere Robot in World3 (see Fig. 2). The average total execution time in this world was around 1.2 s. One problem with the motion planner in this case is that the initial configuration and goal configuration are in terms of Euclidean distance extremely close to one another, but there is an impassible obstacle between them. Whenever an attempt is made to connect a configuration to the goal configuration, selecting the nearest configuration is actually not the right thing to do. The motion planner does not find a solution until a random configuration is generated that is closer to the goal configuration than the initial configuration, which takes quite a long time. CONNECT cannot exploit its ability to connect nodes to the goal quickly, but incurs a penalty by trying this connection until a configuration passed the wall has been found.

The performance of the CONNECT and EXTEND algorithms are quite similar in World3. CONNECT and EXTEND do not handle the case where the initial configuration and goal configuration are very close but an impassible obstacle is blocking the shortest path between them. A potential solution to this case is discussed in Sect. 5.

The selection of the bias probability affects the motion planner's ability to explore unexplored areas or converge to the goal configuration. A bias probability that is too high will not explore and similarly a bias probability that is too low will take a long time to converge to the goal configuration. We found that a bias probability between 10 and 40 percent produced the best results for the humanoid robot, as shown in Fig. 6. When the bias probability is too high or too low the total execution time goes up. A balance between exploration and bias is required to achieve optimal results for a wide variety of motion planning problems. As the bias probability is increased the number of random

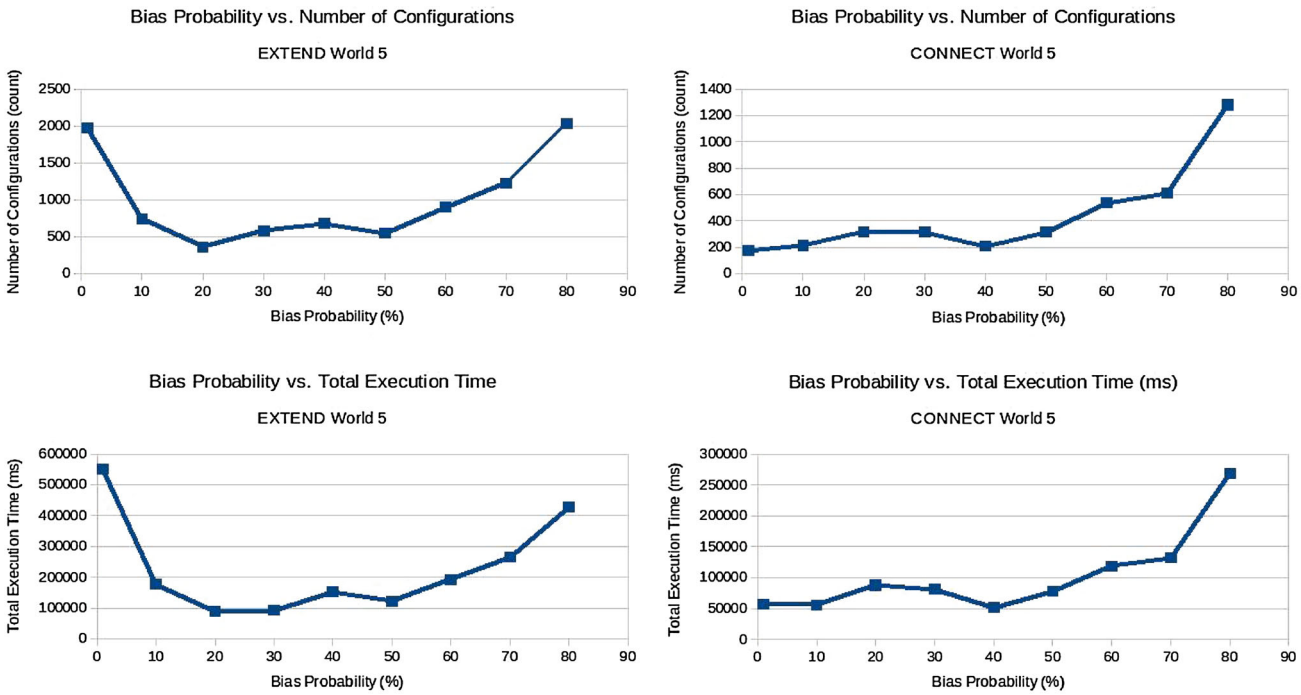

Fig. 6 Comparison of EXTEND and CONNECT-RRT in humanoid robot blitz domain

configurations follows. However, the additional configurations provide little to no useful additional information since the majority of samples are biased the dispersion is low (i.e. there is little exploration).

The previous experiments showed the practicality and efficiency of our motion planning system. We focus on the performance of our system in the humanoid robot environments. Our system proved efficient and versatile. It was able to solve problems in about 1.23 s as shown in Fig. 6 for the humanoid robot.

One obvious important aspect of any motion planner is whether or not the robot can actually execute the generated plan. By qualitative inspection and the real world experiment with the humanoid robot, there is no doubt that the sphere and humanoid robot would be able to successfully execute the motion plans generated by our motion planner.

5 Conclusions and Future Work

There are a number of possible optimizations and improvements that could be performed in our implementation. One possibility is to always pick the goal on the first iteration of the RRT construction algorithm. This handles the simple cases where there is a direct collision-free path to the goal configuration from the initial configuration.

Another simple improvement that could be implemented is, regardless of bias probability, to choose the goal configuration if the last configuration added to the RRT is within distance ϵ of the goal configuration. This could potentially be a way to prevent unnecessary additional exploration when a configuration in the RRT is already close to the goal configuration and thus likely to have a direct collision-free path to the goal.

One problem with our motion planner identified in experiments with the Sphere Robot in World3 is when the initial configuration and goal configuration are close to one another but there is no direct path between the two configurations. A potential solution to this problem could be to try to connect every configuration added to the RRT to the goal configuration. Unfortunately, since the current move test implementation uses the incremental simulator stepped in real-time, move tests are slow. This optimization would result in a significant increase in total execution time. If the move test implementation is improved in the future, then this optimization could be implemented.

A different solution to this problem is to use the k-NN instead of just the NN ([19] use a similar approach but for different reasons). To understand why using k-NN would be advantageous, consider the case where the path from the nearest neighbor to the goal is blocked, but another close neighbor has a direct collision-free path. In that case, if a connection to that neighbor is tried, the path can be found much faster. Of course, there is a trade-off when using k-NN. If move tests are an expensive operation and all of the k-NN are blocked, the additional move test time is k times longer.

References

1. Berenson D, Srinivasa S (2010) Probabilistically complete planning with end-effector pose constraints. In: Proceedings of ICRA
2. Geraerts R, Overmars MH (2002) A comparative study of probabilistic roadmap planners. In: Workshop on the Algorithmic Foundations of, Robotics, pp 43–57
3. Hsu D, Latombe J-C, Kurniawati H (2007) On the probabilistic foundations of probabilistic roadmap planning. In: Thrun S, Brooks R, Durrant-Whyte H (eds) Robotics research. Springer tracts in advanced robotics, vol 28. Springer, Berlin, Heidelberg, pp 83–97
4. Jaillet L, Simeon T (2004) A PRM-based motion planner for dynamically changing environments. In: Proceedings of IROS 2004, vol 2, pp 1606–1611
5. Karaman S, Frazzoli E (2011) Sampling-based algorithms for optimal motion planning. Int J Rob Res 30(7):846–894
6. Kavraki LE, Latombe JC, Motwani R, Raghavan P (1995) Randomized query processing in robot path planning (extended abstract). J Comput Syst Sci:353–362
7. Kavraki LE, Svestka P, Latombe J-C, Overmars MH (1996) Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Trans Robot Autom 12(4):566–580
8. Kuffner Jr. JJ, Kagami S, Nishiwaki K, Inaba M, Inoue H (2002) Dynamically-stable motion planning for humanoid robots. Auton Robot 12(1):105–118
9. Kuffner Jr. JJ, LaValle SM (2000) RRT-Connect: An efficient approach to single-query path planning. In: Proc. IEEE Intl Conf. on Robotics and Automation, pp 995–1001
10. LaValle SM (1998) Rapidly-exploring random trees: A new tool for path planning. Technical report, Carnegie Mellon University
11. LaValle SM, Kuffner Jr. JJ (2000) Rapidly-exploring random trees: progress and prospects. In: Algorithmic and Computational Robotics: New Directions, pp 293–308
12. Lindemann SR, LaValle SM (2004) Incrementally reducing dispersion by increasing voronoi bias in RRTs. In: Proceedings of ICRA 2004, vol 4, pp 3251–3257
13. Lozano-Perez T (1980) Spatial planning: a configuration space approach
14. Reif JH (1979) Complexity of the mover's problem and generalizations. In: SFCS '79: Proceedings of the 20th Annual Symposium on Foundations of Computer Science. IEEE Computer Society, Washington, DC, pp 421–427
15. Selig JM (1992) Introductory robotics. Prentice hall, USA
16. Spong MW, Hutchinson S, Vidyasagar M (2005) Robot modeling and control, chapter 3. Wiley, New York, pp 73–110
17. Sucan I, Kavraki LE (2012) A sampling-based tree planner for systems with complex dynamics. IEEE Trans Robot 28(1):116–131
18. Tsianos KI, Sucan IA, Kavraki LE (2007) Sampling-based robot motion planning: towards realistic applications. Comput Sci Rev 1:2–11
19. Urmson C, Simmons R (2003) Approaches for heuristically biasing RRT growth. In: Proceedings of IROS