Single-Query Robot Motion Planning using Rapidly Exploring Random Trees

M.Sc. Thesis Proposal

Jonathan Bagot (6801879)

Supervisor: Dr. Jacky Baltes

April 2, 2012

Department of Computer Science

University of Manitoba

Winnipeg, Canada R3T2N2

bagotj, jacky@cs.umanitoba.ca

ABSTRACT

The research to be conducted tackles the general movers problem in the context of motion planning for robots. The proposed system solves the general movers problem using a sample based tree planner combined with an incremental simulator. A motion simulator shall be developed for rapid prototyping of robot and world models where the results of the planner can be transferred to a real robot, logged, and visualized. The performance of the proposed planner shall be evaluated with data from simulation and real world experiments.

1. PROBLEM STATEMENT

Motion planning algorithms determine continuous collision free paths from an initial position to goal position. The motion plan is constrained by the physical characteristics of the robot and the environment. Robots with many degrees of freedom (DOF), redundant manipulators, and real-time requirements increase the difficulty of the motion planning problem. The goal of my thesis is to develop a planner able to consider kinodynamic constraints, find or approximate the inverse kinematics (IK) solution, and perform collision detection quickly and reliably in order to generate feasible motions plans.

2. INTRODUCTION

Motion planning is done by humans every day and typically without much thought. Imagine a simple task you perform every morning such as making a cup of coffee. Making a cup of coffee requires many motion plans. For example consider the first

task to get a cup from the cupboard. You must first walk over to the cupboard and position your body such that the distance from your shoulder to the cupboard handle is less than your arm length. The required sequence of movements for your body to achieve this position from your initial position can be called a motion plan. A good motion plan would be quick, require little effort, and avoid any part of your body bumping into obstacles in the kitchen. For each subsequent movement a motion plan is required. In order to complete the task you would determine a motion plan to move your hand to grasp the cupboard handle, move your hand to open the cupboard door, move your hand to grasp the cup, and so forth. You may even decide to use both hands; one to open and close the cupboard door and the other to grasp the cup. For humans this can be called a simple task, but in reality there are many complex systems at work. Determining motion plans is a difficult problem for an autonomous robot that has equivalent sensory and movement capabilities in comparison to a human.

The motion planning problem for robots is a form of the general mover's problem an extension of the classical mover's problem also known as the piano mover's problem. The piano mover's problem is to determine how to move a piano from one point to another without collision. 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". A robot would have many masses linked together by joints. A DOF for a robot is a controllable joint that defines the configuration of the robot. The number of DOF defines the dimension of the moving object. The general movers problem was shown to be PSPACE-hard by Reif [10]. A problem that is PSPACE-hard can be solved with a Turing machine with a polynomial amount of memory and unlimited time. As the number of DOF increases the runtime follows for deterministic solutions.

Since the motion planning problem for robots is PSPACE-hard, finding paths quickly for robots with large DOF in uncertain environments requires sacrificing planner optimality for query response time. A path that is good enough but not optimal allows for fast think and react cycle time. Reducing the think and react cycle time is a critical task for robots that interact with the real world. Deterministic complete planners are well defined but are known to be intractable for robots with large DOF and real-time requirements in even simple domains. Sample based probabilistically complete planners (will find solution if one exists but cannot determine if one does not exist) have showed promise returning sub-optimal results in reasonable time. Sample based planners can be classified in two general groups: roadmap and tree [14]. 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 an answer. The first learning phase builds the map and the second query phase determines the plan with a graph search algorithm such as Dijkstra's algorithm. The learning phase is an expensive operation but in a static environment the roadmap can be used for all queries, therefore the cost of the learning phase is only incurred once. In a dynamic environment the cost of the learning phase would be too large to react to changes in the environment since the roadmap must be rebuilt before each query. 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 start state and the tree is built incrementally until the end state is reached. The answer to the query is simply a branch of the tree from the root (start) to a leaf (end) node, therefore the answer is found in a single query. Building the tree is not an expensive operation which makes tree planners ideal for dynamic environments where plans may become invalid quickly. The focus of the research to be conducted is on sample based tree planners because the goal is

to solve the motion planning problem in dynamic environments with real-time requirements.

The increasing complexity of robots has brought on a whole new set of challenges for robotics research. The cost of building small scale robots has decreased considerably, which facilitates active research in robotics. An increase in the number of DOF and a desire for autonomy in uncertain environments with real-time requirements leaves much room for improvement in the current popular motion planning algorithms.

The proposed thesis work makes many contributions to the motion planning problem for robots such as using a sample based tree algorithm combined with an incremental simulator, real world application, and an extensive analysis of the different elements that constitute a sample based tree algorithm; namely the Rapidly Exploring Random Tree (RRT).

By combining a sample based tree algorithm with an incremental simulator, the motion planner not only generates a plan but tests the feasibility of each step in real-time as it is generated using a model of the robot and world. The plan can then be executed on a real robot with greater confidence that the plan is valid.

Many other solutions to the motion planning problem for robots have been proposed but have only been demonstrated in simulation. For example variations of sample based tree planners were proposed in [5, 6, 3, 7, 17] but were only evaluated in simulation with models that are always dynamically stable (cannot fall over) and fictitious environments. The proposed thesis work shall be demonstrated in both simulation and the real world. Not all solutions translate well from simulation to real world, therefore application to real world is necessary to show the effectiveness of the solution.

A RRT is a data structure for planning problems where the nodes of the tree are generated randomly and typically biased towards un-explored areas. There are many elements of a RRT that can be implemented in various ways. Different implementations behave and perform better or worse for specific problems. The proposed thesis work shall not use a single implementation of a RRT but instead use many combinations of different element implementations in order to analyze how each behaves and performs for specific types of problems. The key elements of a RRT are sample bias, nearest neighbor, distance metric, and collision detection. Each element effects the performance of the RRT in different ways. For example the sample bias method has a direct impact on how well the tree covers the environment while the nearest neighbor method only limits how fast random configurations can be connected to the tree.

3. BACKGROUND

Motion planning for robots encompass many difficult problems. Finding a path may mean solving the IK problem, performing collision detection, and considering the kinodynamics of the robot to ensure the plan is feasible.

The IK problem for a robot manipulator is to find the joint angles given the goal position of the end effector. There is no general closed form solution to the IK problem for robot manipulators with greater than 6 DOF [12]. Finding a solution to the IK problem requires solving many non-linear equations simultaneously, which is a difficult task in mathematics. In fact, manipulators are often designed with simplification of the IK problem in mind so that techniques such as kinematic decoupling can be used. Kinematic decoupling allows the position and orientation of the end effector to be solved separately.

Collision detection provides a method to determine if two objects intersect. The execution time of collision detection

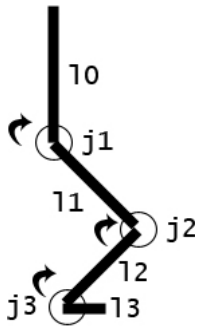

Figure 1: 4 link kinematic model of humanoid robot.

algorithms vary depending on the number of dimensions and the resolution required.

Kinodynamic constraints refer to velocity and acceleration limitations which are primarily determined by joint properties. Without considering these constraints the plan will not work on real hardware because in reality there are no motors that can exert an infinite amount of force.

An essential ingredient to solving planning problems is a unique understanding of the state space. The state space consists of every possible combination of inputs and the respective outputs. For a large number of inputs the size of the state space yields a combinatorial explosion of states. Finding the optimal solution would require searching through the state space for a set of states which arrive at the goal state from the initial state and maximize the heuristic function. For example if a robot has a limited power supply, the heuristic function might rank solutions that use less power higher. In general searching through the entire state space is infeasible for large state spaces especially if a solution must be found in real-time. A solution might be considered acceptable if the power consumption does not exceed a fixed threshold although it may not be the optimal solution.

The most commonly used state space representation for robot motion planning is known as a configuration space (CSPACE) [8] introduced by Perez. 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. CFREE denotes the set of all robot configurations that are not in collision with obstacles. COBS is the compliment of CFREE, therefore COBS 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.

For single kinematic chain robots with a fixed base (non-mobile) and a CSPACE where COBS is an empty set, the path planning problem is reduced to solving the IK problem. Given the goal position and orientation of the end effector, the IK problem is to determine the joint angles. On the contrary the forward kinematics (FK) problem is to determine the position and orientation of the end effector given the joint angles.

A simple robot model consists of links connected by joints as depicted in Figure 1. In order to solve the FK problem a coordinate frame must be attached to each link as depicted in Figure 2. Coordinate frame $o_i x_i y_i z_i$ will be attached to link i and the coordinate frame of l_0 is the base frame. There is also a homogeneous transformation matrix T_j^i which represents

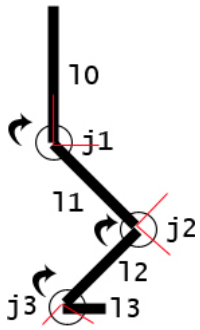

Figure 2: 4 link kinematic model of humanoid robot with coordinate frames attached.

the position and orientation of frame j relative to frame i. Although it is possible to attach the coordinate frames arbitrarily and still find the correct solution, it is better to have a standard convention of selecting coordinate frames that allows for comparison of results between peers. The Denavit Hartenberg (D-H) convention is a commonly used method for selecting frames of reference. Coordinate frames are selected using two rules. The first rule states that the x-axis x_i is perpendicular to the z-axis z_{i-1}; the second rule states that the x-axis x_i intersects z-axis z_{i-1}. Spong et al. [12] show that four parameters (link length a_i, link twist α_i, link offset d_i, and joint angle θ_i) are enough to represent the homogeneous transformation matrix when these rules are satisfied. Once each homogeneous transformation matrix is known it is possible to calculate the position and orientation of coordinate frame i relative to the base frame by right multiplying homogeneous transformation matrices $(T_0, ..., T_{i-1})$ where T_i is defined by Equation 1 and 2 [12]. If only revolute joints and two dimensional coordinates are considered as in the simple model shown in Figure 1 Equation 2 is reduced to Equation 3. The last column of transformation matrix T_i are the coordinates for coordinate frame i relative to the base frame, or simply put the coordinates of the end effector.

$$Rot_{z,\theta_i} Trans_{z,d_i} Trans_{x,a_i} Rot_{x,\alpha_i} = T_i =$$

$$\begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

$$T_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

$$T_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i & a_i\sin\theta_i \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

The IK problem on the other hand may not always have a solution, or a unique solution. Even if a solution exists, it may be hard to obtain. A common approach to solving the IK problem is kinematic decoupling which considers the position and orientation of the end effector separately. Kinematic decoupling is only possible if the design of the kinematic chain supports position and orientation separation. An example of such a design is a robot manipulator with a spherical wrist. There is no general closed form solution to the IK problem for robot manipulators with greather than 6 DOF, therefore numerical solutions or hill climbing methods are used.

Physical systems fall into two groups holonomic and non-holonomic. In a holonomic system, the controllable DOF are equal to the total DOF. In a non-holonomic system, the controllable DOF are less than the total DOF. The configuration of a non-holonomic robot is dependent on the path taken to achieve the configuration. Redundant manipulators have more DOF than the dimension of the work space position vector, but can be either holonomic or non-holonomic. Having a redundant manipulator can present more challenges when solving the IK problem because of singularities, which are caused my multiple configurations that produce the same end effector configuration.

The Nearest Neighbor (NN) search problem in CSPACE is defined as follows; given a set S of configurations and a specific configuration p in the set, the nearest neighbor problem is to determine the closest configuration q to p. Naive approaches calculate the distance between every configuration in set. For a large set of configurations this is not feasible if the NN must be computed in real-time. The naive approach has a linear execution time although there are known approaches that execute in near logarithmic time. An effective NN search algorithm is critical for both roadmap and tree based planners. The most popular NN search methods use kd-trees to solve the problem which usually perform better than other search methods in euclidean space [16]. For roadmap planners connecting vertices requires an NN search method, similarly tree planners must connect a random configuration to the nearest configuration in the tree. The NN search is used many times during the planner execution, therefore the NN search method must be efficient.

4. RELATED WORK

Sample based roadmap planners such as probabilistic roadmaps (PRM) [4] introduced by Kavraki et. al. must first build a map (learning phase) then search for a solution (query phase). 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. The learning phase creates the PRM by randomly generating configurations in CFREE which are connected to nearby configurations. The underlying data structure of a PRM is an undirected graph where nodes represent configurations and edges represent paths. The initial and goal configuration are inputs to the query phase. The graph can be efficiently traversed using already well defined methods in graph theory such as the A* search algorithm. One advantage of using the

PRM is that there are very few parameters and heuristics required for implementation. A major disadvantage of using the PRM is the number of connections that must be made between configurations which is not a simple task. 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.

Cell decomposition methods such as Binary Space Partitioning (BSP) break down the configuration space into free and blocked cells recursively starting with one large cell that is considered to be mixed. Entropy can be used to intelligently select partition points that generate better partitions and smaller trees than standard partitioning methods [2]. The information gain is determined by the relative proportion of free and blocked areas. Once the free cells are known, a planner can use the tree output from the flexible BSP algorithm to generate a collision free path. A major disadvantage of BSP is that it does not directly facilitate the CSPACE representation. In order to generate CFREE, configurations would have to be generated that lie within the free cells. For this reason, this approach would not be suitable for robots with large DOF and real-time requirements.

La Valle introduced Rapidly-Exploring Random Trees (RRT) [5, 6] a data structure for planning problems with non-holonomic and differential constraints. A RRT is a tree composed of quasi uniformly distributed random configurations rooted at the initial configuration of the robot. 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 set CFREE. The goal is to generate a leaf node which is the goal configuration. The RRT algorithm terminates once the goal configuration is achieved, therefore no unnecessary work is done. A RRT does not require a learning phase and the output translates directly to the CSPACE representation. In a single query, the plan is known which makes the RRT data structure a viable option for robot motion planning in dynamic environments with real-time requirements.

Constructing a RRT requires a few fundamental components. At each iteration of the algorithm a random configuration is generated. It is important that the RRT explore un-explored areas. To do this it is necessary for there to be a method to bias random configurations towards the unexplored areas (sample bias method). Each random configuration is connected to the existing tree by selecting the nearest configuration in the tree (nearest neighbor method). To do this a distance metric d such as euclidean distance must defined to measure the distance between configurations (distance metric method). It is also important that the path between two configurations lies completely in CFREE or in order words is collision free (collision detection method). Pseudo code [5] with comments for a RRT in the context of a CSPACE is given in Algorithm 1.

RRTs share many of the same benefits as PRMs among being easy to implement but also have many notable advantages. For example, expansion of the RRT is biased towards unexplored regions of state space and the distribution of nodes is consistently close to sampling distribution. The RRT is always connected and has minimal edges. The most notable advantage of a RRT is the fact that they do not require connections between configuration pairs which is an expensive operation in building a PRM.

Since the introduction of RRTs there have been many adaptations to improve the performance of the data structure for

Algorithm 1 RRT

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.
    $near_c \leftarrow NearestNeighbor(rand_c, RRT)$ ▷ Uses kd-tree.
    $motion \leftarrow Move(rand_c, near_c)$ ▷ Motion that minimizes distance between two configurations.
    $new_c \leftarrow NewConfiguration(near_c, motion, \delta t)$ ▷ Configuration generated by motion from $near_c$.
    $RRT.addConfiguration(new_c)$ ▷ Adds configuration to the tree.
    $RRT.addEdge(near_c, new_c, motion)$ ▷ Path between two configurations when motion is performed.
  end for
  return RRT
end procedure

different planning problems. RRT-Connect [3] generates two trees, one tree is grown from the initial configuration and the second tree is grown from the goal configuration. The extend heuristic is replaced by the connect heuristic which repeatedly extends multiple steps until the other tree or an obstacle is hit. Multipartite RRT (MP-RRT) [17] generates a forest of disconnected trees and uses a sample bias method based on information from previous configurations in trees that belong to the forest.

A goal of the research to be conducted is to combine the RRT with an incremental simulator and optimize the individual components of the RRT for problems with real-time requirements.

An incremental simulator computes the current state after a set of inputs has been applied over a period of time. Given state $x(t)$ a function is defined which solves $x(t + \delta t)$. Recently Sucan et. al. [13] have successfully combined sample based tree planners with a physics engine. The physics engine was used as an incremental simulator to handle kinodynamic constraints and compute state $x(t)$. A similar approach shall be used in the research to be conducted, but unlike Sucan et. al. the sample based tree planner is a RRT, and complex robots in many different environments will be demonstrated.

5. PROBLEM DESCRIPTION

Quickly determine feasible path between initial configuration and goal configuration that satisfy the robot and environment constraints using a coarse model of the robot. Use path to develop a motion plan that translates well to a real robot.

6. SOLUTION STRATEGY

Develop a motion planner using a single-query sample based tree algorithm in the context of a CSPACE. The motion planner will build a RRT while exploring the CSPACE. Nodes in the tree will represent robot configurations. Robot configurations will be randomly generated and used if they meet the following intelligent criteria, they belong to CFREE, the robot configuration is dynamically stable, and the robot configuration is feasible given realistic joint limitations (no joints that can exert infinite force). The RRT creation will be optimized for real-time applications by only using methods that can be executed quickly. The methods that have the fastest execution time will be determined empirically by investigating the runtime impact of different

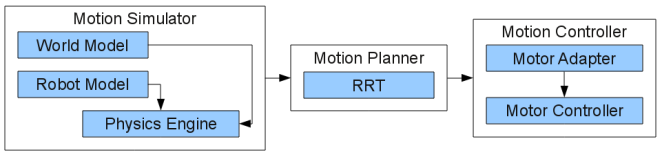

Figure 3: Solution Strategy

nearest neighbor methods, sample biasing methods, incremental distance algorithms, and distance metrics. The performance of the motion planner shall be compared to a simple BSP path planner as a baseline.

For the NN search method, space partitioning algorithms will be compared to the naive approach. The naive approach compares the query point to every other point and keeps track of the nearest, therefore it has a worst case complexity of $O(n)$ where n is the number of points. Space partitioning algorithms such as those that use a kd-tree data structure have an average complexity of $O(logn)$. Although it may seem that there is no point to compare the naive approach to a space partitioning algorithm, the naive approach uses less memory because no complex data structure is required. The naive approach can outperform space partitioning algorithms as the number of dimensions increases [15]. Two distance metrics will be used, euclidean distance and a fast euclidean distance approximation. Approximate Nearest Neighbor (ANN) is an expandable C++ library developed at the University of Maryland [9] that is free for use under GNU LGPL. ANN will be used to quickly compare and possibly customize different NN search methods because implementing all methods from scratch is simply not feasible for the scope of this thesis.

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 [7]. 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 [6] 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. As described in section 4 the BSP algorithm uses entropy to aid in partition point selection; this research will investigate the use of mixed goal and entropy bias in a RRT.

Computing state $x(t)$ at time t is achieved by using a physics engine as an incremental simulator. The output of the physics engine is used as an input to the motion planner. The physics engine shall be used to test the feasibility of robot configurations given kinodynamic constraints, find solutions to the IK problem, and perform collision detection.

6.1. Physics Engine

An incremental simulator is required to test each configuration in the RRT at time t. The incremental simulator should use collision detection to confirm that a configuration belongs to CFREE, kinematics to confirm that a configuration satisfies joint limitations, physics calculations to confirm that a configuration is dynamically stable, and kinodynamic constraints are satisfied.

Open Dynamics Engine (ODE) is an articulated rigid body simulator [11] that is fast, robust, stable, and highly config-

urable. ODE allows configuration of parameters that effect dynamics such as gravity and friction coefficients. Kinodynamic constraints can also be enforced on joints. At every time step collision detection is performed which produces a list of contact points. A contact point represents the intersection of two objects in the world. While ODE provides built in collision detection, custom collision detection can be integrated into the physics engine. The IK problem is not solved by ODE but hill climbing is used to determine if joint configurations meet the goal configuration while satisfying joint limitations.

6.2. World Model

The world model consists of simple objects such as cubes, cylinders, planes, and spheres combined together to generate complex environments. The forces that act upon these objects can be simulated using the physics engine; forces such as friction and gravity. The motion simulator user interface will provide an easy method to create, modify, and arrange objects in the world.

6.3. Robot Model

Robots can be modelled [12] as a set of joints connected by links. There are many different types of joints, but only revolute joints shall be used. This is without loss of generality because it is not difficult to model other joints in a similar manner. Revolute joints rotate on a single axis, therefore each revolute joint represents a single degree of freedom (DOF). Each link has an associated length and connects two joints, unless a kinematic chain ends with a joint there is always one more link than there are joints.

6.4. Motion Simulator

Develop a tool for rapidly prototyping robot and world models using C++, QT, OpenGL and ODE. The motion simulator can be used to visualize the CSPACE and RRT in either real-time or faster than real-time as the motion planner executes. The motion simulator will simplify debugging and conducting experiments with different motion planner parameters.

The minimum requirements for the motion simulator are as follows:

- Create, save, and load world models.
- Create, save, and load robot models.
- Define initial and goal configuration.
- Configure motion planner parameters.
- Run simulations in batch mode.
- Toggle visualization mode on and off.
- Generate and record data relevant for evaluation.
- Integration with motor adapter for real world testing.

6.5. Motor Adapter

A motor adapter converts robot model joint angles to motor positions with a function $f(\theta)$. Each motor type requires a different motor adapter. ODE supports many different joint types including ball, hinge, slider, contact, universal, fixed, and angular. Revolute joints shall be modelled with angular motors in Euler mode, and constrained to rotation on a single axis with appropriate motor parameters high stop, low stop, velocity, and max force.

Define $f(\theta)$ for each motor type by using motor documentation. For example with a Robotis AX-12 Dynamixel motor 2^{10} positions can be specified where each position maps to an angle between [0, 300]$^\circ$ therefore the mapping function is a simple linear equation 4.

$$f(\theta) = \theta * (2^{10}/300) \tag{4}$$

6.6. Motor Controller

The motor controller contains the logic necessary to move n motors simultaneously when motor commands are received. Motor commands provide a high level interface for motor position specification. The motor controller includes Pulse Width Modulation (PWM), trajectory control, interpolation, Proportional Integral Derivative (PID) control, and serial communication. The output from each motor adapter will be sent to the motor controller.

Use existing Robotis AX-12 motor controller developed for FIRA 2008 and RoboCup 2008 [1]. Extend existing motor controller if additional Robotis motor types are available for use.

7. EVALUATION

7.1. Simulation

Four robot models shall be built using the Motion Simulator described in section 6.4. Three of the four robot models will be built for real world testing. The first simple model is for proof of concept; a single 3-dimensional point with 3 DOF for velocity along each axis. The point model will be used in worlds with various static and dynamic clutter. The second model represents the classic Selective Compliance Articulated Robot Arm SCARA a commonly used model for assembly robots. The SCARA has a fixed base and 4 DOF, it can reach any 3d coordinate in it's workspace. The thrid model will be representative of an industrial 6 DOF manipulator with a fixed base, contrary to the SCARA model the end effector yaw, pitch, and roll can be controlled. The second and third manipulator models shall be used to demonstrate planning for grasping and moving tasks in worlds with static clutter. The fourth model will be of a bi-pedal robot with 19 DOF that is representative of a humanoid robot Blitz which has competed at FIRA and RobotCup 2008. For all robot models each DOF must be considered by the planner in order to achieve the goal position. For example the planner for a walking gait of a bi-pedal robot must still consider the position of the arms in order to maintain balance.

Various static and dynamic world models shall be created using the Motion Simulator described in section 6.4. The

moving point model will be tested in worlds with randomly generated obstacles. The manipulator and humanoid models will be tested in worlds with static obstacles and blocks to grasp and move. The static obstacles and blocks will be replicated in the real world.

7.2. Real World

Three robots shall be built using AX-12 motors. The first two robots will be a small scale 4 DOF SCARA and 6 DOF manipulator, the manipulators will be designed using the models described in section 7.1. The third robot Blitz is a humanoid robot with 19 DOF built for FIRA and RobotCup 2008. The purpose of building real robots and using them in the real world is to verify the validity of the simulation.

7.3. Data Gathering and Performance Criteria

The following data shall be gathered for analysis:

- Planner RRT construction time.
- Planner RRT query time.
- Planner execution time.
- Planner memory footprint.
- Size of the RRT, number of configurations in tree.
- Dispersion [7] of the RRT. A measurement of how well the CSPACE is covered by the RRT.
- CPU load. Tasks that can be done in parallel will be identified in order to optimize CPU usage.
- Quality of plan in simulation compared to real world by visual inspection. Can the real robot execute the plan? Does the real robot achieve the goal configuration from the initial configuration collision free?

7.4. Optimization

Using the data gathered described in section 7.3, another iteration of the motion planner shall be developed. The optimizations implemented will be determined after analysis of the data and based on the bottlenecks that are found.

8. RESOURCES REQUIRED

The robots used for experiments and evaluation of the proposed solution require a Robotis Bioloid Kit which includes AX12 motors, linkage for the joints, and an Atmel ATMEGA128 processor for the motor controller. The kit will be available for use in the Autonomous Agents Laboratory.

9. TIMELINE

Table 1: Timeline

Task	Start Date	End Date
Literature Review	January, 2008	N/A
Implementation	April 2012	July 2012
Experiment	July 2012	October 2012
Evaluation of Results	October 2012	December 2012
Write Thesis	December 2012	April 2013

10. SUMMARY

The motion planning problem for robots is a difficult problem. There remains much room for improvement in the current popular motion planning algorithms. The proposed thesis work makes many contributions to motion planning for robots including but not limited to, using a sample based tree algorithm combined with an incremental simulator, real world considerations and application, and an extensive analysis of the impact of different methods in RRTs.

11. REFERENCES

[1] BAGOT, J., ANDERSON, J., AND BALTES, J. Vision-based multi-agent slam for humanoid robots. In *Proceedings of the 5th International Conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS-2008)* (June 2008), pp. 171–176.

[2] BALTES, J., AND ANDERSON, J. Flexible binary space partitioning for robotic rescue. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Las Vegas, October 2003), pp. 3144–3149.

[3] JR., J. J. K., AND LAVALLE, S. M. Rrt-connect: An effcient approach to single-query path planning. In *In Proc. IEEE Int. Conf. Robot. Autom. (ICRA)* (2000), pp. 995–1001.

[4] KAVRAKI, L. E., SVESTKA, P., LATOMBE, J.-C., AND OVERMARS, M. H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. on Robotics and Automation 12*, 4 (1996), 566–580.

[5] LAVALLE, S. M. Rapidly-exploring random trees: A new tool for path planning. Tech. rep., 1998.

[6] LAVALLE, S. M., KUFFNER, J. J., AND JR. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions* (2000), pp. 293–308.

[7] LINDEMANN, S. R., AND LAVALLE, S. M. Incrementally reducing dispersion by increasing voronoi bias in rrts. In *In Proc. IEEE Int. Conf. Robot. Autom. (ICRA)* (2004), vol. 4, pp. 3251–3257.

[8] LOZANO-PEREZ, T. Spatial planning: A configuration space approach, 1980.

[9] MOUNT, D. M. *ANN Programming Manual*, 2010.

[10] REIF, J. H. Complexity of the mover's problem and generalizations. In *SFCS '79: Proceedings of the 20th Annual Symposium on Foundations of Computer Science* (Washington, DC, USA, 1979), IEEE Computer Society, pp. 421–427.

[11] SMITH, R. *Open Dynamics Engine v0.5 User Guide*, February 2006.

[12] SPONG, M. W., HUTCHINSON, S., AND VIDYASAGAR, M. *Robot Modeling and Control*. Wiley, 2005, ch. 3, pp. 73–110.

[13] SUCAN, I. A., KRUSE, J. F., YIM, M., AND KAVRAKI, L. E. Kinodynamic motion planning with hardware demonstrations. In *IROS* (2008), pp. 1661–1666.

[14] TSIANOS, K. I., SUCAN, I. A., AND KAVRAKI, L. E. Sampling-based robot motion planning: Towards realistic applications. *Computer Science Review 1* (August 2007), 2–11.

[15] WEBER, R., SCHEK, H.-J., AND BLOTT, S. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA* (1998), A. Gupta, O. Shmueli, and J. Widom, Eds., Morgan Kaufmann, pp. 194–205.

[16] YERSHOVA, A., AND LAVALLE, S. M. Improving motion-planning algorithms by efficient nearest-neighbor searching. *IEEE Transactions on Robotics 23*, 1 (2007), 151–157.

[17] ZUCKER, M., KUFFNER, J., AND BRANICKY, M. Multipartite rrts for rapid replanning in dynamic environments. *Robotics and Automation, 2007 IEEE International Conference on* (April 2007), 1603–1609.