Path and Motion Planning of Point Mass Robots

GROUP 12

Tanay Rastogi Niklas Rolleberg

1991-07-21 1993-06-15

tanay@kth.se nrol@kth.se

Abstract

The task of finding a path between two states in an environment with obstacles for a robot with kinodynamic constraints is nontrivial. We present a solution to this problem which can be used for any motion model. The solution is based on RRT* and shares the same asymptotically optimal characteristics and is able to deliver competitive results while remaining easy to adapt to new motion models.

Contents

1 Introduction

For any robotics research and development, the ultimate goal is to have autonomous robots. The robots which will accept a high level instruction and then perform the assigned task without any human intervention. The robots has to decide on its own how to perform the task. One of such task can be path planning.

Path planning is the problem of finding a path, given starting point and goal point, in an environment that contains obstacles so that the robot does not collide with any of them [6]. Path planning is a non-trivial problem in robotics. Any action in the physical world is constrained by the law of physics, dynamic changes in the world, uncertainty in the information about the world and geometric constrains. While solving the problem, the robot has to take care of all constrains to plan its path in the free space. One of the classical path planning problem is called Piano movers problem, where one tries to find how to move a piano in a CAD model of the house. The algorithm has to give the answer to how the piano should be moved from one room to another by not hitting anything in the house. Here the problem completely ignore the dynamic constrains of the piano movement and solve the problem just in terms of translational and rotational motion of piano. The actual robot motion is more complex than the piano mover problem. In most cases, the robot motion problem is constrained by two things constrains in the environment and constrains in the robot own motion. Environment can be bounded or un-bounded and can be clustered with obstacles which are either static or dynamic. Also, the planning depends a lot on the amount of information known about the environment, if the position of the obstacles are known or not in advance of planning. The robot is itself is constrained by it dimensions, geometry, and constrain on its motion. The path found by the differential drive robot might be infeasible for car like robot because of constrain on steering angle for the car. The problem of path planning which we are trying to solve here, can be described as: [6],

- Let A be a single point mass object, that is, the robot moving in the Euclidean space W, called workspace, represented by R_N with N = 2.
- Let O_1, O_2, O_Q be fixed object distributed in W. They are called obstacles.
- Assuming that the geometry and location of the O_1, O_2, O_Q are known.
- The kinematic constrains on the motion of A is known.

- The problem is - Given initial position and orientation and goal position and orientation of A in W, generate a path P, specifying a continuous sequence of positions and orientation of A avoiding contact with O_1, O_2, O_Q , starting at initial position and orientation and ending at the goal position and orientation.
- Report failure if no such path occurs.

1.1 Contribution

For the purpose of solving the motion problem, we have used different motion models, each having different kinodynamic constrains on the motion for the point mass. In order to find the path between the start state and the goal state, there is an incremental sampling based planner called **informed Rapidly - exploring Random Tree star (informed RRT*)**. The planner solves the path planning problem in high dimensional state in reasonable computation time. In RRT the planner tries to find the path by expanding from the start and then keep on extending from there towards the goal point. The RRT is an incremental simulator where the search tree containing the parent and child. RRT randomly finds a state in the free space and then the nearest child node tries to reach the random node, using the differential constrains on its motion. The point reached will become the new child and the previous child will now be parent. The algorithm keeps on doing the same in all free space until the goal is reached. Once the goal position is found, then the algorithm tries to optimize the path fond by checking if there is another path which can reduce the cost for traversing the path. The optimization is further improved by including heuristics constrain to optimize only within area bound by the ellipse, which is formed by the start and goal as foci and then reducing the diagonals as the better path is found. The informed-RRT* finds the most optimal solution with probability 1, when the algorithm runs for the reasonable computational time .

As per the problem statement, we want to reach to the goal point, and the point is not just the position but also the orientation and velocity (the state). The informed RRT* can only help us to reach the near the position of the goal, but most of time does not able to reach to the correct orientation and velocity.

1.2 Outline

In this report we present a solution to the path planning problem specified in Assignment 1. In Related work we write about the state of the art. We then

present our solution in Method. Images and performance data of our method are presented in experimental results, these results are then discussed and compared to other solutions in Summary and conclusions.

2 Related work

In order to do path planning in the given free space, there are numerous methods to achieve this. It can be solved by solving the path problem in continuous space or by discretizing the space via cell decomposition or Voronoi methods and then apply graph search algorithm. Visibility graphs can be used to find the path in continuous space directly. For discrete space, there are different approaches to attain the optimal path, like blind, Dijkstra, A*or D*. These algorithms are often resolution complete and resolution optimal. They guarantee to find the optimal solution, if solution exits[3]. All the above algorithms works for holonomic robots, and have no differential constrains in the motion. These algorithms cannot be expanded to higher state space, anything more than 2D or 3D. Hence, need for the algorithms that can be used in higher dimensions and are able to handle differential constrains to find the optimal path. Another way to find the path is through the random sampling of the state space and increment the search stochastically. Stochastic searches, like RRT and PRM, use sampling based methods to avoid the need for discretization. They are better in taking care of the higher dimensional state space and can involve differential constrain while planning. These algorithms are guaranteed to of finding the optimal path once time goes to infinity[4]. For the problem statement given, we have used the expansion of the RRT algorithm, called informed RRT*. The RRT algorithm builds on ideas from optimal control theory, non-holonomic planning, and randomize path. The basic idea is to use control theoretic representations and incrementally grow a search tree from an initial state by applying control inputs over time to reach new state[5]. The RRT is an incremental simulator where the search tree containing the parent and child. Even though they are better in solving the path planning, when compared to PRM or graph based, they suffer from several issues. One of the RRT is that it does not find the optimal path. The path found by the algorithm is always random and does not consider any heuristic to find the path[5]. An extension of RRT called RRT* improves upon that. The RRT* tries to optimize the path by finding better connections that minimize the cost between the child and parent node. The RRT* are asymptotically optimal, improvement on the vanilla RRT[4]. An extension of the RRT* can even be used for motion with differential constrains, called kinodynamic RRT*. The algorithm focus on the motion

model to move towards the goal point, such that the robot reaches with the correct state, of velocity and orientation[1]. Another problem that RRT, and the RRT*, suffers that they tries to find the optimal path by traversing in complete free space. This approach increase the computational time to reach the optimal solution. The informed RRT* solves the problem by limit the search to a sub-problem that would have a better solution[3]. The informed RRT* runs a sub-problem once the traditional RRT* search the path. The algorithm then creates an ellipse with goal and start position as the foci, such that the area bounded by the ellipse contains the path found. Then algorithm tries to reduce the area of optimization by reducing the diagonals of the ellipse. The kinodynamic RRT*, discussed by [1], can make sure that the robot reaches in the correct state or orientation and velocity towards the goal. But still we found that the algorithm was not able to be implemented in all motion model. More the complex system is, difficult it become to apply kinodynamic RRT*. Hence in order to reach in correct position and state, we have used several method for motion models to reach the goal with desired states.

3 Method

In section we describe how we solved the problem and how we implemented it.

3.1 Basic algorithm

The solution we chose to use for solving the problem was the Rapidly exploring random tree (RRT). We wanted use the same algorithm for all models so we had to choose a method which was able to account for the specific constraints for all models and still be able to find a close to optimal solution. We used the informed RRT* [3] to search the state space for the goal.

In order to expand the RRT in a way that is compatible with the motion models, we used model specific functions for expanding the tree to a new random point and find a path between two states. The expansion to a random point is used when adding new branches to the tree, and since the position of the random points are randomized it is not necessary to reach the exact point. When trying to connect two states however it is important to end up at the correct place with the correct velocity, this requires a more advanced method. Further explanation of how these methods work can be found under the next topic. Before a new branch is added to the tree it is checked for collisions to make sure it is feasible, if the path isn't feasible it is rejected

and the algorithm continues without adding the new branch to the tree.

3.2 Model specific

Since all the models are slightly different we had to come up with several different ways of building the RRT.

3.2.1 T1-kinematic point

The kinematic point is controlled by speed in x and y direction.

Towards random point. Since the kinematic point can change velocity instantaneously all that is needed is to calculate the direction towards the random point and apply the maximum velocity towards it, the algorithm then calculates the points position after randomized time, this point is then considered as the point reached.

Reaching an exact state The direction towards the desired state is calculated, and maximum velocity is applied towards it. When the point has reached the correct position, velocity matching the desired state is applied and the state is reached.

3.2.2 T2-Dynamic point

The dynamic point is controlled by acceleration

Towards random point. Our algorithm applies acceleration perpendicular to the points velocity to make it steer towards the randomized point. The acceleration is recalculated in each time step to prevent a S-shaped path. The algorithm simulates the point for a randomized number of time steps, and the position reached after the simulation is finished is considered the new point.

Reaching an exact state. In order to reach a exact state a more advanced approach is needed. The first thing we must do is to match the speed of the goal state, so maximum acceleration is applied to increase or reduce the speed of the point. Once the correct speed has been reached we calculate the path towards the goal state with Dubins curves [2]. The radius of the circles used by Dubins curves are derived from the circular orbit equation (Equation 1). This gives us the optimal path for this (constant) speed since all turns are taken as sharply as possible, thus minimizing the distance traveled.

$$a = \frac{v^2}{r} \tag{1}$$

3.2.3 T3-differential drive

The differential drive model is controlled by a rotation speed and speed in the forward direction.

Towards random point. We decided to use a greedy approach to find the path to a random point. Our algorithm calculates the states for several control inputs in each time step and keeps the one which reduced the distance to the point the most, this state is then used as origin for the next time step. The number of time steps are randomized.

Reaching an exact state. Our algorithm starts by setting the speed to 0 and applies maximum rotation speed to aim the model towards the target state. Once the model is pointing in the right direction maximum velocity is applied until the target state is reached, once the model is at the desired position it stops again and turns towards the desired direction. Finally the desired velocity is applied and the exact state is reached.

3.2.4 T4-kinematic car model

The kinematic car is controlled by speed and a steering angle.

Towards random point. The model is simulated over a randomized number of time steps. In each time step a range of steering inputs are evaluated based on how much they reduce the distance to the goal point. The state reached by the best steering angle is used as origin for the next time step. The function for calculating the next state is shown in Algorithm 1. The speed of the car does not impact the controllability of the car, so it is set as high as possible.

Reaching an exact state. When trying to reach the desired state our algorithm uses the same method as when steering towards a random point, but with one difference. Instead of steering towards the point we want to model to end up at it steers towards a point on the tangent line of that point. The point on the tangent line is placed slightly in front of the orthogonal projection of the model on the tangent line (see figure 1), and as the model

Algorithm 1 Calculate next state

```
1: procedure NEXTSTATE(a, b)
2:     bestState ← Null
3:     for steeringAngle ∈ −maxSteer : maxSteer do
4:         newState ← Evaluate(steeringAngle)
5:         if newState < bestState then
6:             bestState ← newState
       return bestState
```

gets closer to the goal, the point also moves along the line. The resulting path is illustrated in figure 2.

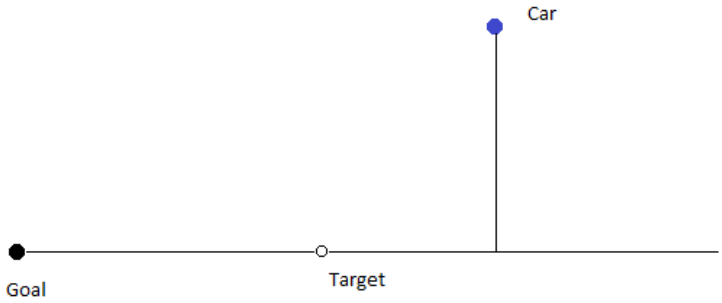

Figure 1: Target point on the tangent line of the goal

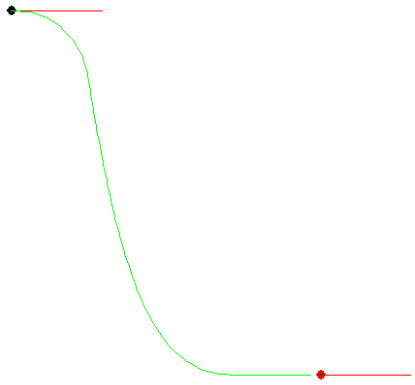

Figure 2: Path of kinetic car

3.2.5 T5-dynamic car model

The dynamic car was controlled in the same way as the kinematic car, except it does not travel at maximum velocity all the time. Since the speed of the

model only can be changed by altering the acceleration, and if the maximum acceleration is small, slowing down in time might be hard to do. Our algorithm slows down/ speeds up to the desired speed once it reaches a certain distance from the goal.

3.2.6 T6-dynamic car model with friction

The dynamic car with friction is almost identical to the dynamic car except for the friction constrains so our algorithm controls it in the same way when going towards a new random point. But when trying to reach an exact state we used dubins curves to calculate the path, much like the dynamic point.

Reaching an exact state. The first thing our algorithm does is matching the velocity of the target state by applying acceleration. The reason for this is to simplify the calculations of the turning angle when doing dubins curves. When the minimum turning radius is calculated our algorithm calculates the dubin curves for reaching the desired state, this is then traversed with constant speed.

3.3 Implementation

We chose to implement the planning algorithm and visualization as two separate programs. The planning algorithm reads the map and calculates the path, then outputs the path to a file which is then used a input for the visualization program. The visualization was done in the virtual robotics platform V-REP. We did however not use the built in physics engine but rather just moved an object small steps to make it look like the the object was behaving like a physical object.

4 Experimental results

In this section we present results from our algorithm.

4.1 T1-Kinematic point

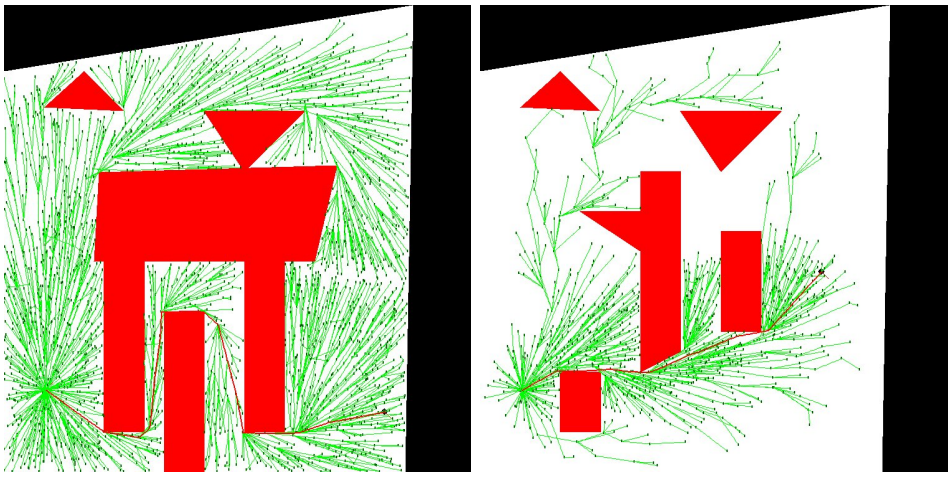

Figure 3: informed RRT* with kinematic point model

4.2 T2-Dynamic point

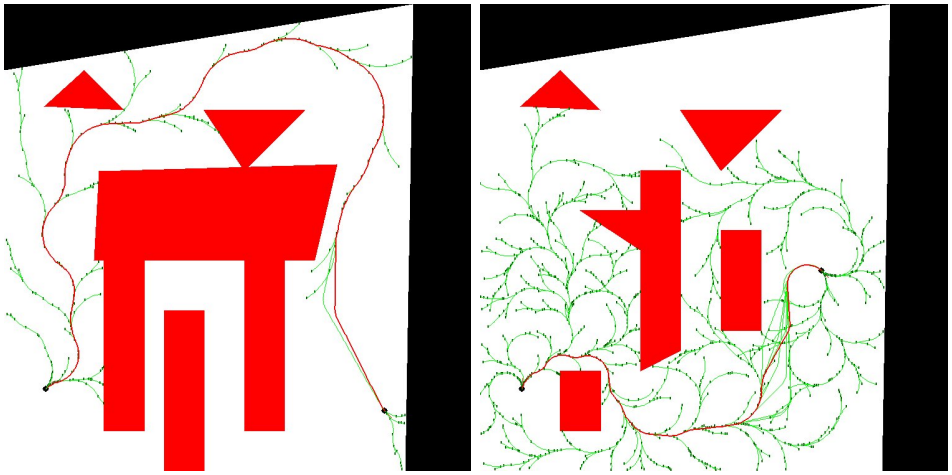

Figure 4: informed RRT* with dynamic point model

4.3 T3-Differential drive

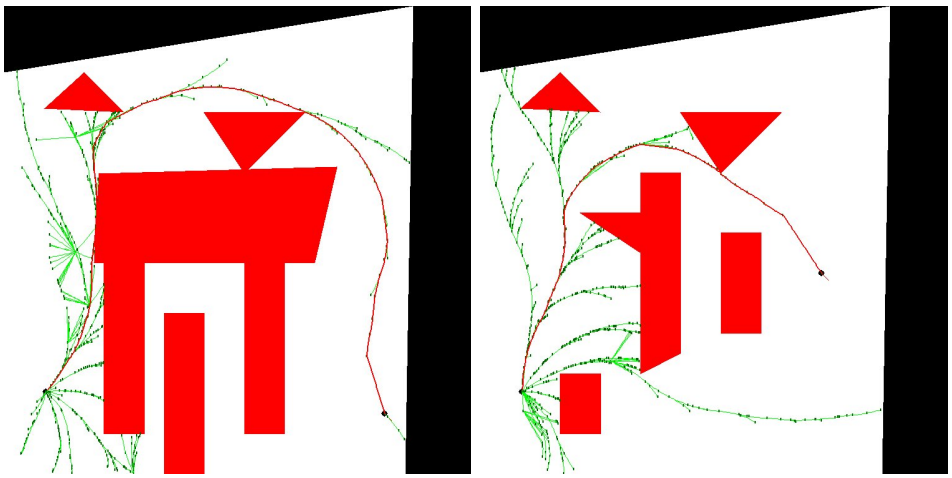

Figure 5: informed RRT* with differential drive model

4.4 T4-Kinematic car

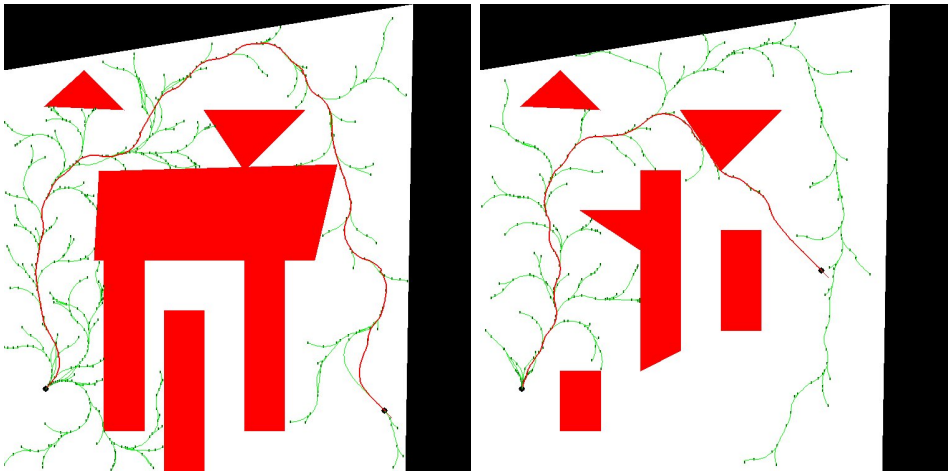

Figure 6: informed RRT* with kinematic car model

4.5 T5-Dynamic car

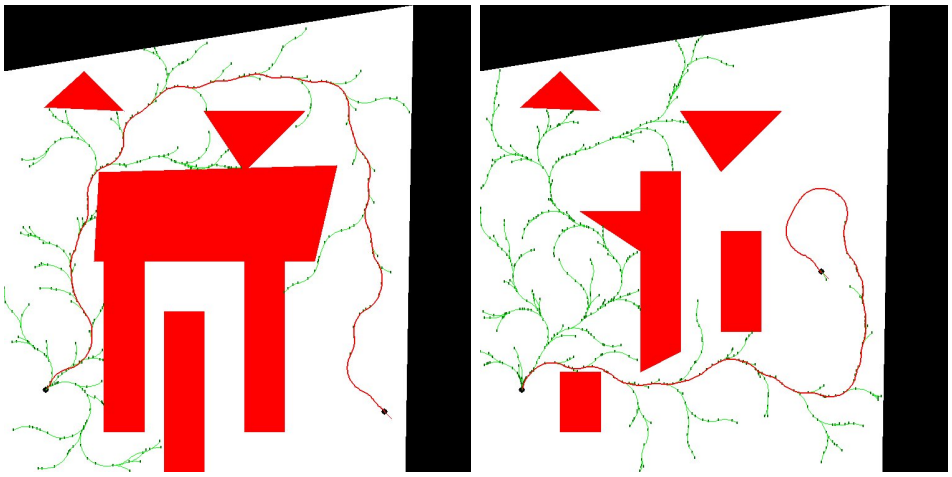

Figure 7: informed RRT* with dynamic car model

4.6 T6-Dynamic car with friction

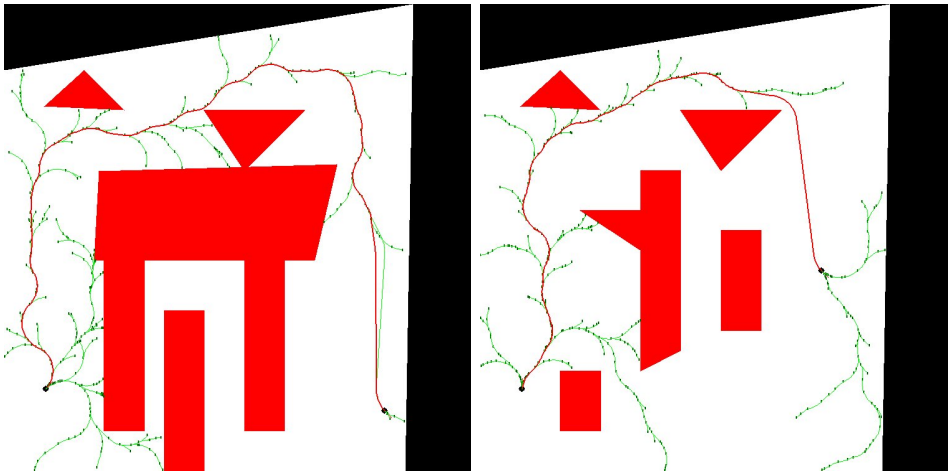

Figure 8: informed RRT* for the dynamic car model with friction

Table 1: Results

	D1	D2	D3	D4	D5	D6	E1	E2	E3	E4	E5	E6
G1	-	-	-	12.07	-	-	-	-	62.81	-	-	-
G2	8.3	19	18.25	21	42.15	31.25	19.25	38.05	50.4	57.8	98.7	86.95
G3	-	-	-	-	-	-	-	-	-	-	-	-
G4	8.1	18.9	-	-	-	-	-	34	-	-	-	-
G5	8.4	15	51.4	-	36.6	-	19.8	64	49.9	-	-	-
G6	8.77	11	18.87	9.25	9.24	-	20.5	-	-	-	23.16	-
G7	-	-	-	-	-	-	-	-	-	-	-	-
G8	8.1	28.85	22.35	-	-	-	18.88	-	50.03	-	-	-
G9	14.8	16.07	-	-	-	-	33.1	-	-	-	-	-
G10	8.25	9.07	17.6	20.8	15.6	21.2	19.5	17.8	-	-	-	-
G11	8.09	8.35	17.8	8.2	8.25	8.25	18.83	20.7	82	19.4	19.3	19.3
G12	**8.33**	**13.29**	**39.32**	**13.6**	**28.1**	**19.7**	**19.97**	**33.6**	**102.8**	**34.7**	**44.7**	**38.54**
G13	8.11	9.957	19.85	11.22	14.7	13.38	18.88	24	44.25	23.21	28.89	30.98
G14	8.14	13.29	20.75	-	-	-	18.9	27	50.1	-	-	-
G15	-	-	-	-	-	-	-	-	-	-	-	-
Best	8.09	8.35	17.6	8.2	8.25	8.25	18.83	17.8	44.25	19.4	19.3	19.3
	G11	G11	G10	G11	G11	G11	G11	G10	G13	G11	G11	G11

5 Summary and Conclusions

From looking at the results in table 1 it is clear that our algorithm does not perform as well as some of the other groups. We believe this is partly because of how long we let our algorithm run. Since it is based upon informed RRT* which does find the optimal path with probability 1, our algorithm should also be capable of this. However the time needed for adding new nodes in the tree increases rapidly when the number of nodes becomes large. Because of this we stopped the algorithm shortly after a path was found. The path improves slowly if the algorithm keeps running.

Our results are similar to other groups which used similar approaches. Group 13 used RRT* and dubins curves to find the path and our results show similar trends as theirs on all problems. Except for the differential drive where our greedy approach to steering is far from optimal. The method used in the car models however seams to work fairly good. The results for the car models are comparable to the results of group 10 who also used RRT* but with a Linear Quadratic Regulator (LQR) for controlling the motion model. The steering method we used (explained in Section 3) is much simpler than LQR since all you do is to try different steering angles and sees which one was the best, and it is also a robust method which would work even if you would

introduce some noise in the model or environment. This steering method did however not work well for the car with friction, and we had to resort to the dubins curves instead. The dubins curves will most likely provide a faster path in most cases, but it pushes the model to the limit to get a path as good as possible thus making it more susceptible to noise.

We think our algorithm does a good job of solving the problem. While the results are not the best, some of them are fairly close the the best times, and considering that our algorithm works for any motion model which can go towards a state and reach a state without obstacles makes our algorithm easy to use and competitive.

References

[1] Jur van den Berg Dustin J. Webb. Kinodynamic rrt*: Optimal motion planning for systems with linear differential constraints. *CONFERENCE - Algorithmic and Computational Robotics: New Directions*, 2000.

[2] Devin J. Balkcom Matthew T. Mason Hamidreza Chitsaz, Steven M. LaValle. Minimum wheel-rotation paths for differential-drive mobile robots. *IEEE International Conference on Robotics and Automation*, 5 2006.

[3] Timothy D. Barfoot Jonathan D. Gammell, Siddhartha S. Srinivasa. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. *The name of the journal*, 11 2014.

[4] Emilio Frazzoli Sertac Karaman. Sampling-based algorithms for optimal motion planning. *Cornell university Library*, 5 2011.

[5] James Kauffner Steven M. laValle. Rapidly exploring random tree: Progress and prospects. *CONFERENCE - Algorithmic and Computational Robotics: New Directions*, 2000.

[6] Bang Wang. A study of mobile robot motion planning. 1994-12-01.