A Motion Planner Based on the Prediction Using LSTM Autoencoder

Xinchao Song

song.xin@northeastern.edu

December 9, 2019

Abstract

The objective of this project is to find a motion planning approach based on predictive information. We propose a motion planer based on rapidly-exploring random graph (RRG) algorithm using future data predicted from a Long short-term memory (LSTM) autoencoder. We construct a maze problem as a simple domain to test our method. Experiments are performed in the simulation and the efficiency of our planning algorithm can be shown and verified.

1 Introduction

As everyone may experience in daily life, it is usually better for us to make a plan and determine an acting policy based on future predictions than the current situation, especially when we are in a dynamic environment with moving objects/obstacles. For example, a running back in an American football game may need to predict the motion track of his rival to try to avoid him. Therefore, it is fascinating if we can find a way to let an intelligent agent also able to act based on motion prediction.

In general, a prediction problem is to estimate a sequence of data describing a future event based on a sequence of data representing the current state, which is usually called sequence-to-sequence prediction problems. It is an important but also challenging topic in the machine learning field. In this project, we will specifically address a motion prediction problem for time series data with an artificial neural network model called LSTM autoencoder and use it to develop a motion planner based on a rapidly-exploring random graph (RRG) algorithm.

For the whole code and some experiment videos, please visit the GitHub page of this project: `https://github.com/xinchaosong/lstm-predictive-motion-planner`.

2 Background

In this section, we will introduce some background of LSTM autoencoder, RRG, and other relevant concepts. We also present some previous work related to our topic.

2.1 Long Short-Term Memory

Long short-term memory (LSTM) is a particular type of recurrent neural network (RNN) architecture [1] widely used for processing entire sequences of data. It is usually composed of a cell and three special structures called "gates", including an input gate, an output gate, and a forget gate, where the cell is to track the relationship between the elements in the input sequence and the gates are a way to let information into and out of the cell optionally. For the three types of gates, as their names suggest, the input gate is to determine the how much new data flows into the cell, the forget gate is to determine how much data should be remembered in the cell, and the output gate is to determine how much data flows out of the cell.

2.2 LSTM Autoencoder

An autoencoder is an unsupervised learning neural network to learn a representation of the input data and attempt to copy it to the output [2]. It consists of two components: an encoder that maps the input into the representation, and a decoder that maps the representation to a reconstruction of the original input. There are several different types of autoencoders popularly used recently, such as Variational autoencoder (VAE) and LSTM autoencoder.

An LSTM autoencoder is an implementation of an autoencoder for sequence data using an Encoder-Decoder LSTM architecture. In this architecture, an encoder reads the input sequence step-by-step and refines it as an internal learned representation for the decoder. LSTM autoencoder usually has an impressive performance on processing time series sequence data, and that is why we use it in this project.

2.3 Motion Planning in Configuration Space

In a motion planning problem, it is very common to solve it in so-called configuration space. Here the term configuration is to describe the pose of the agent in the environment. Accordingly, configuration space is a vector space that contains all possible configurations of the agent. It is usually a high-dimensional space, and its dimension is the minimum number of parameters needed to specify the configuration of the robot completely (i.e., degrees of freedom, DOFs). The advantage to using configuration space is that it can convert the original planning problem into a planning problem for a single point and therefore makes the planning process simpler and more straightforward.

2.4 Rapidly-Exploring Random Tree/Graph (RRT/RRG)

A rapidly exploring random tree (RRT) is an algorithm for solving path planning problems. [3] It will aggressively probe and explore the search configuration space by randomly and incrementally building a space-filling tree from an initial configuration to find a path that reaches the goal configuration. However, one of the disadvantages of RRT is that it does not guarantee to find optimal paths. One method to overcome this problem is a variant of RRT called Rapidly-Exploring Random Graph (RRG), which is constructing a graph rather than a tree to explore the space and can converge towards an optimal solution by using the shortest path algorithms such as breadth-first search.

2.5 Previous Work

One of the early and widely cited applications of the LSTM autoencoder was [4], where an LSTM autoencoder model was proposed to both reconstruct and predict sequences of frames of input video data as an unsupervised learning task. More applications of LSTM autoencoder on motion prediction have also been made. [5] proposes a music-oriented dance choreography synthesis method using LSTM autoencoder to extract a mapping between acoustic and motion features. [6] develops an LSTM autoencoder network for vehicle trajectory prediction, which can generate the future trajectory sequence of surrounding vehicles in real-time. [7] builds a multi-layer LSTM autoencoder network that predicts future frames for a robot navigating in a dynamic environment with moving obstacles. However, the previous work usually focuses more on the motion prediction only rather than learning an acting policy based on that motion, which is what we plan to do in this project. In the perspective of motion planning in an environment with moving obstacles, most of the previous work is mainly dealing with short-term planning by using real-time observation data, such as [8] and [9], rather than a long-term planning based on predication

3 Method

In this section, we will propose a generalized model for motion planning to let the agent move without any collision in an environment that contains moving obstacles. Our model consists of two components, an LSTM autoencoder, and a motion planning algorithm based on RRG.

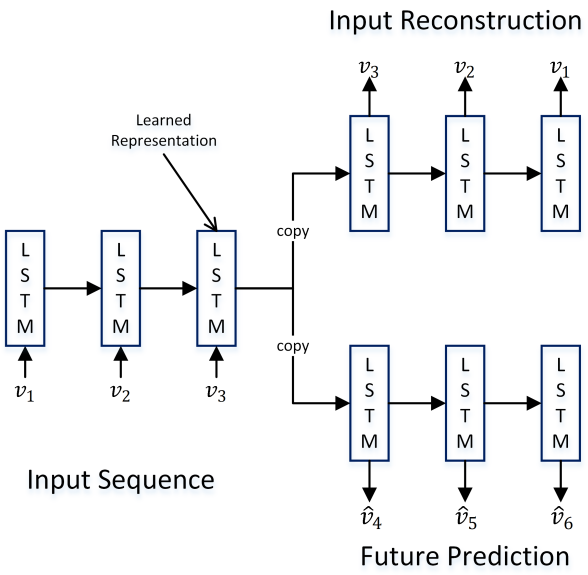


Figure 1: The composite LSTM autoencoder model

3.1 LSTM Autoencoder

The LSTM autoencoder model following the method proposed by [4] will be built and trained to perform motion prediction. In essence, there are three components in the model, namely Encoder, Decoder, and Predictor, which are shown in Figure 1. Encoder projects the input sequence into a dense vector in latent space, capturing the semantic information v_t of the input sequence, which is, in turn, decoded by Decoder back into sequence. Predictor, on the other hand, predicts future sequence information $\hat{v}_t$ up to time steps T_p. The model is trained via Stochastic Gradient Descent and backpropagation through time by using the mean squared error loss function

$$L = E[(V_{1:T} - V'_{1:T})^2] + E[(\hat{V}_{T+1:T+n} - \hat{V}'_{T+1:T+n})^2].$$

where the first term is the mean squared error between the input and reconstructed sequence, while the second term is the mean squared error between the ground truth sequence and the predicted sequence.

Generally, the lengths of all three sequences will be the same such as 50:50:50, although the predicted future sequence does not have to be the same length as the input sequence. Therefore, here, we will use a different sequence in our experiment so that we can predict a farther future, and the ratio between our three sequences will be 50:50:100.

3.2 RRG Planner

Algorithm 1 RRG Based on Prediction

initialize the experiment environment
Collect the input data sequence D through observation
Find the future trajectory of the moving obstacle t by using prediction
while a valid path is found **do**
  Initialize the configuration space, the origin node q_0 and the node graph G
  while the goal q_g is not reached **do**
    Select a valid configuration point q_{rand}
    Find the nearest node q_{near} a valid configuration node q_{rand}
    Temporarily add the edge (q_{near}, q_{rand}) to G
    Find the shortest reaching time s from q_0 to q_{rand}
    if q_{rand} collides with any moving obstacles at s **then**
      remove the edge (q_{near}, q_{rand}) from G
      continue
    end if
  end while
  Find the shortest path p from from q_0 to q_{rand}
  if any node in p collides with any moving obstacles **then**
    continue
  end if
end while
Find the final policy according to p

Basically, the process preform RRG planning using the prediction from LSTM autoencoder can be separated in three steps as follows.

1. Collect input data from the environment
2. Make a prediction based on input data to predict the trajectories of all moving obstacles
3. Perform RRG based on the predicted positions of the moving obstacles and find a policy

Based on this process, we can implement the planning algorithm shown in Algorithm 1.

3.3 Evaluation Strategy

Different evaluation strategies will be used in this project. First, the effectiveness of motion prediction using the LSTM autoencoder will be verified by the loss on the data set as well as by the visual measurement. Secondly, the whole planning algorithm will be tested in simulation, and the corresponding performances will be presented and evaluated. Third, a baseline approach will also be discussed as a contrast to show the effectiveness of our method.

4 Experiment and Result

In this section, we will first describe a maze problem for testing our algorithm as well as the experiment environment and the data set we are using. Then the training result of LSTM autoencoder based on the data we collected for the maze problem will be shown. Finally, we will present the performance of our motion planner in the simulation.

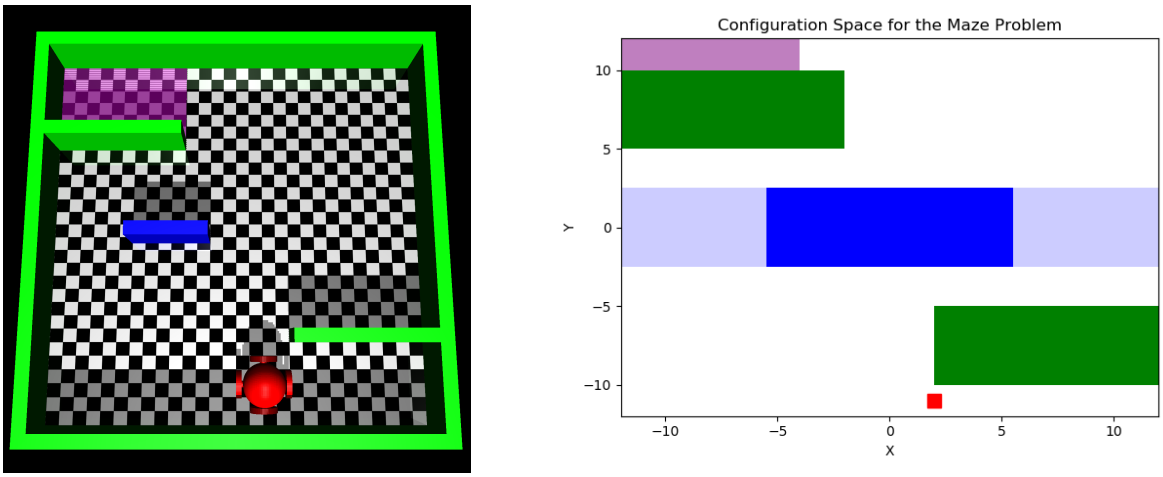
4.1 The Maze Problem

Considering the limited time for this project, here we propose a simple "maze" problem with only one moving obstacles to verify our idea. The layout of the maze is shown in Figure 2. As we can see, it is a wall-surrounded area where contains a robot in red and a center moving obstacle in blue. We can also construct the configuration space for this problem shown in Figure 2b, where the red dot is the robot, the blue block is the configuration of the center moving obstacle with a light blue area showing its moving range, the green blocks are the configurations of the two inner walls, and the purple area is our target area.

In this environment, the center moving obstacle will keep moving left and right in the range $[-10, 10]$, and its position along the x axis is the observation. The robot can be controlled to move up, down, left, and right. The planning goal is to move the robot to reach the purple target area without collision with the center moving obstacle.

4.2 Simulation Environment and Data Source

We use the MuJoCo physics simulator for implementing the maze problem, collecting data, and perform the experiments. The maze constructed in MuJoCo is shown in Figure 2a. Same as the conceptual model, the cart can be controlled to move by an input action command 0 or 1 to the simulator, and the angle of the



(a) The maze in MuJoCo

(b) The configuration space of the maze

Figure 2: The layout of the maze

probe can be outputted from the simulator.

We collect all data from the MuJoCo simulation to generate our data set. The starting position and direction of the center moving obstacle are random. The unit of the data set is a trajectory, i.e., a sequence of the x positions that the center obstacle moves from the starting time to end time. In every trajectory, we set the running time of each step to be 0.5 seconds and the maximum number of steps to be 150. To improve efficiency, we will use multithreading to perform the collection. The size of the data set using in the following sections is 110,000, where 100,000 (91%) of it will be used for training, 10,000 (9%) for the validation/test.

4.3 Model Training

The LSTM autoencoder will be trained by using PyTorch. The reconstruction and prediction losses during the training and validation are shown in 3, where we can see that our LSTM autoencoder converges and the mean squared errors are almost 0 after 40 epochs. To guarantee the accuracy of our prediction, we will continue to train our model until 600 epochs.

We can verify the reconstruction and prediction accuracy be comparing the trajectories generated from the given data with the ground truth. Some result examples are shown in Figure 4. We can visually observe that our model can reconstruct the original data and predict future data very accurately, and the effectiveness of our model can be proved.

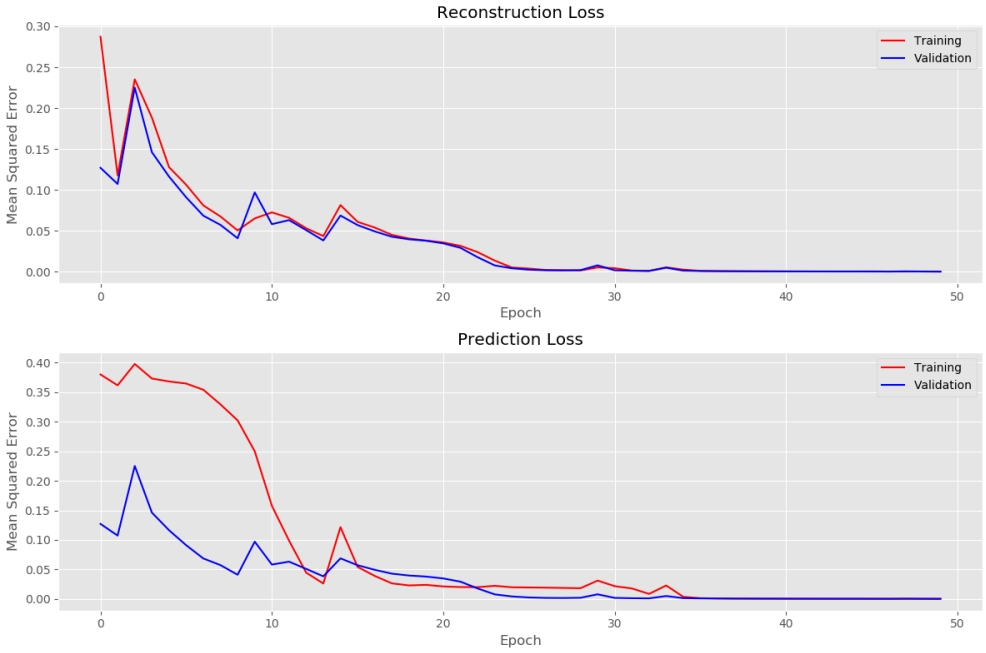


Figure 3: The reconstruction losses and prediction losses for training set and validation/test set against training epochs

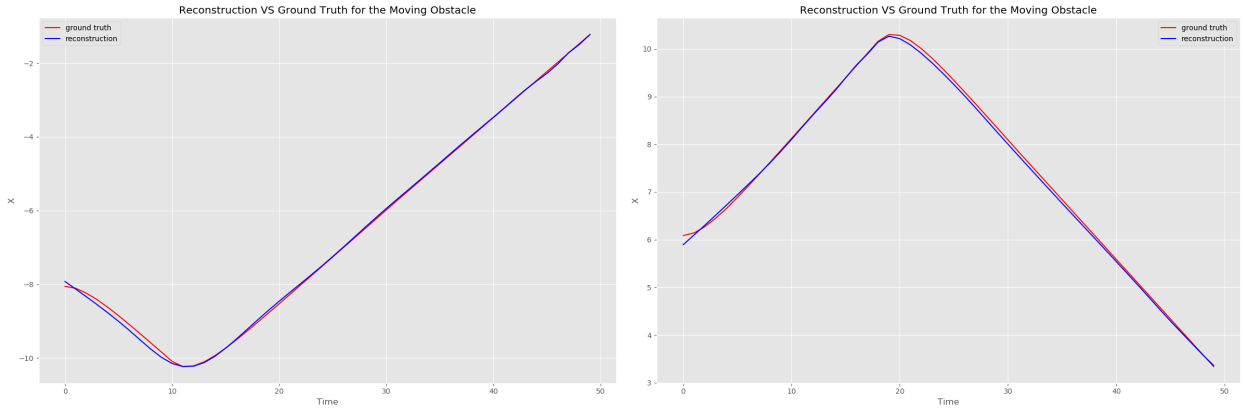
4.4 Motion Planning and Execution in Simulation

Numerous experiments are performed in MuJoCo simulation, and we can always observe that our planner is able to find valid motion policies successfully. An example that shows how the robot devours the center moving obstacle in its path can be seen in Figure 5.

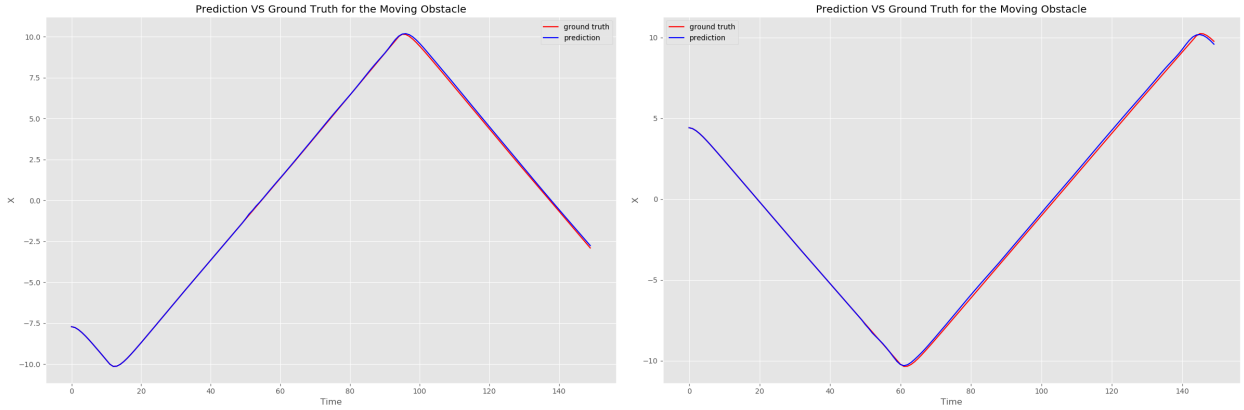
We also tested the original RRG algorithm without prediction as a contrast baseline. The comparison result is shown in Table 1. We can see that our approach has a significantly better performance than a naive RRG planner.

Table 1: The comparison between the baseline RRG and our planner with LSTM autoencoder

	Baseline RRG	RRG with LSTM autoencoder
Success	2	10
Failure	8	0
Total	10	10



(a) Two examples showing the motion trajectories reconstructed by the model and the ground truth



(b) Two examples showing the motion trajectories predicted by the model and the ground truth

Figure 4: The example showing motion trajectories generated by the model (blue) and the corresponding ground truth (red)

5 Conclusions

In this project, we propose a motion planer using LSTM autoencoder to predict future information and RRG to plan a motion policy based on that information. We first introduce the approach of representation learning and future prediction to perform prediction using LSTM autoencoder. Then we develop a planning algorithm using prediction data based on RRG. We also construct a maze problem to test our idea. Several experiments are performed, and the results are presented. The experiment results can firmly show that our LSTM autoencoder can perform a good prediction, and our planning algorithm based on it can find valid motion policies as expected.

References

[1] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[3] Steven M. LaValle. Rapidly-exploring random trees : a new tool for path planning. 1998.

[4] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using lstms. *CoRR*, abs/1502.04681, 2015.

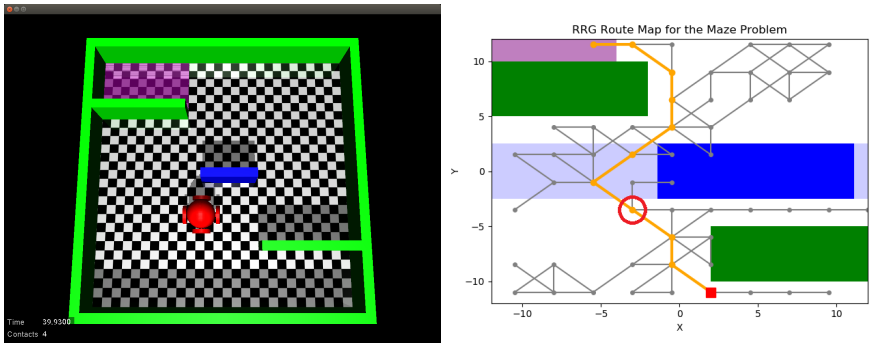
[5] Taoran Tang, Jia Jia, and Mao Hanyang. Dance with melody: An lstm-autoencoder approach to music-oriented dance synthesis. pages 1598–1606, 10 2018.

[6] SeongHyeon Park, Byeongdo Kim, Chang Mook Kang, Chung Choo Chung, and Jun Won Choi. Sequence-to-sequence prediction of vehicle trajectory via LSTM encoder-decoder architecture. *CoRR*, abs/1802.06338, 2018.

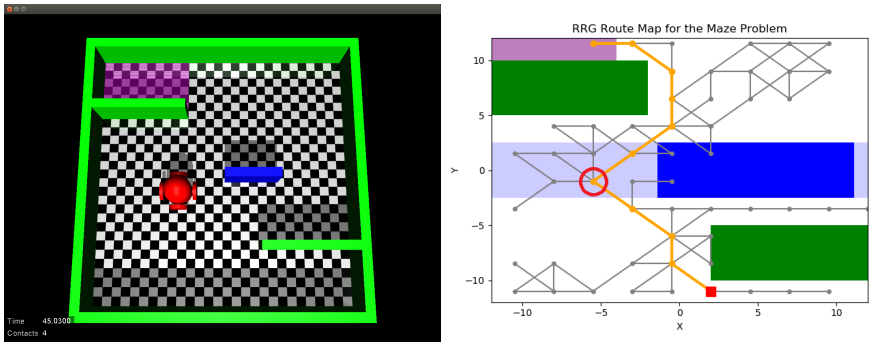
[7] Meenakshi Sarkar and Debasish Ghose. Sequential learning of movement prediction in dynamic environments using LSTM autoencoder. *CoRR*, abs/1810.05394, 2018.

[8] L. G. Torres, A. Kuntz, H. B. Gilbert, P. J. Swaney, R. J. Hendrick, R. J. Webster, and R. Alterovitz. A motion planning approach to automatic obstacle avoidance during concentric tube robot teleoperation. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2361–2367, May 2015.

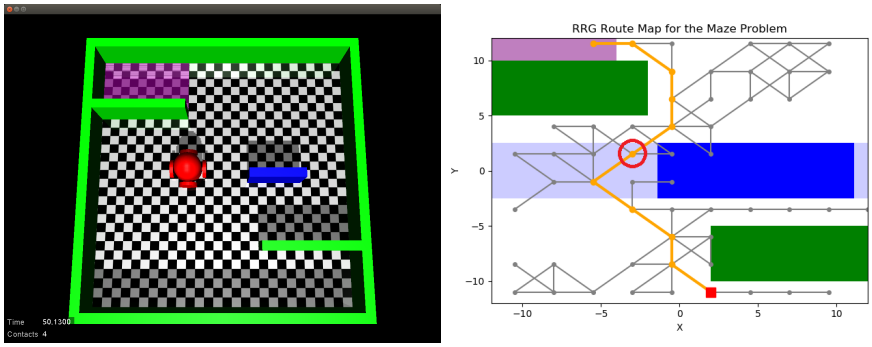
[9] Devin Connell and Hung La. Dynamic path planning and replanning for mobile robots using rrt. pages 1429–1434, 10 2017.



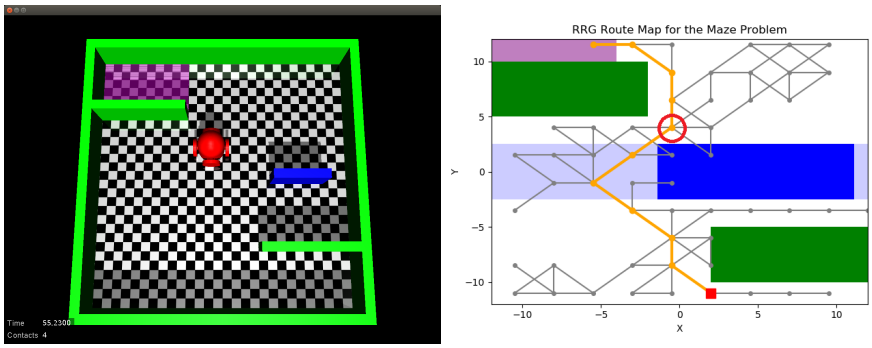
(a)



(b)



(c)



(d)

Figure 5: Four moving steps detouring the obstacle during the execution of the motion plan