SCRAM: Simultaneous Control, Neural Networks, and Mapping for Autonomous Path Planning and Localization

Alexander Schperberg, Kenny Chen, Maggie Xiao, and Vivek Krishnamurthy
Fall 2019, EE209AS: Computational Robotics
University of California, Los Angeles
{*aschperberg28, kennyjchen, vivekkmurthy, zxiao2015*}*@ucla.edu*

***Abstract*—While modern robotic sensing methods excel in their capacity to account for sensor and environmental uncertainties through probabilistic frameworks, many control algorithms assume absolute, error-free states during trajectory planning. Model predictive control (MPC), for example, offers a deterministic formulation during its N-step lookahead and assumes states to be free of error throughout its finite time-horizon; this could affect the efficacy of its next control action if under perturbation from noise or map uncertainties. To this end, we propose a framework which integrates state covariances estimated from a neural network into the MPC loop to provide the optimizer with information of each state's potential uncertainty boundary. The neural network is initially trained offline on data collected from the simultaneous localization and mapping (SLAM) process and then used online to predict the propagation of covariances in new maps during each finite time-horizon optimization of the MPC. The main contributions of this paper are: (1) a framework which predicts a probability distribution associated with each possible future position in the MPC trajectory time-horizon; and (2) we evaluate the feasibility of an online path planning framework that combines three major modalities, namely MPC, Rao-Blackwellized particle filter SLAM, and neural networks for path planning optimization under uncertainty. We validate our method by comparing against a naive approach and show that the neural network allows the MPC to control the robot more aggressively while still ensuring safety.**

***Index Terms*—localization, mapping, SLAM, model predictive control, neural networks, particle filter, uncertainty**

I. Introduction

The field of robotics has made remarkable progress in providing various tools for more robust path planning. Neural networks for example, have been applied to enhance path planning methods as it offers a robust mathematical framework to enable an agent to discover optimal behavior through trial-and-error interactions with its environment (e.g., training). Simultaneous localization and mapping (SLAM) on the other hand consists of concurrent state estimation of an agent via proprioceptive and exteroceptive sensors and model estimation of its environment. These measurements can be used to provide state and map estimates necessary for path planning. Lastly, model predictive control (MPC) provides the capacity for a path planner to generate optimized responses while respecting the robot's dynamic equations and constraints on both control and state variables. These methods, among others, may be implemented to create a new wave of modern path planning architectures. This is significant as new path planners are needed to safely navigate a robot from an initial position to a goal position while accounting for unexplored and uncertain environments.

There are also practical applications to more sophisticated path planners. The most obvious is in search and rescue operations. For example, the goal state could be the last known geographical location of a missing person. Our algorithm would have the potential to find an optimal path to that location without prior knowledge of the environment. Robots with robust vision and path planning capability can also be beneficial to the visually impaired. To be adequate assistants, these robots must plan paths that sufficiently avoids obstacles and accounts for dynamic changes in the environment while protecting itself from harm.

Researchers in the field of collision avoidance may use new path planners to decrease uncertainty in autonomous robots and vehicles. Lastly, surgical robotic research are also currently investigating modified methods of SLAM for image-guided surgery [1]. Thus, new path planning formulations may provide an interesting avenue in combining vision and precise path planning control over surgical instruments.

A. Related Work

There has been much research in developing path planning methods to move a robot from an initial state to a final state. Conventional path planners such as *Dijkstra's* [2] or A* [3] achieve this goal by discretizing a free space into grid cells and applying a search strategy to find the shortest path. However, these discrete path planners are sub-optimal in continuous and high dimensional space for three reasons: 1) optimization is limited by the grid resolution, 2) explicit map representation is required, and 3) handling differential constraints is not possible [4]. Conversely, algorithms such as *Probabilistic Roadmap* (PRM) [5] or *Rapidly-exploring Random Tree* (RRT*) [6] are designed to operate in high dimensional space and can handle both nonholonomic and kinodynamic differential constraints.

Although PRM and RRT* are popular methods used in current robotic research [7] [8], they are insufficient in dealing with uncertainty due to their deterministic formulation. This is problematic as robotic systems must overcome motion, sensing, and environmental uncertainty [9]. One approach to

account for these uncertainties is using a *Partially-Observable Markov Decision Process* (POMDP) [10]. In POMDP, the robot maximizes a reward function that depends on a sequence of future system states and controls in belief space [10]. The drawback of POMDPs is that they are computationally intractable due to the *curse of history* (i.e., complexity increases exponentially with the planning horizon), and the *curse of dimensionality* (i.e., a discrete grid world with n-number of cells corresponds to a n-dimensional continuous space) [11]. Although the computation time is non-trivial even for a small number of states and actions, significant work has been done to reduce the complexity of POMDPs through *point-based iteration* [12] or *automated model approximation* [13].

Still, incorporating an unknown environment in POMDP remains a challenging endeavor, because a changing environment has many possible layouts during execution. The authors in [14] handle this problem by modelling all possibilities within the assumption of a static world, where the model is only updated when new information is acquired by the *Simultaneous Localization and Mapping* process (SLAM) [15] [16]. Although the study successfully implemented POMDP with SLAM in an unknown environment, their online calculations approximated 30 seconds [14]. In dynamic environments this is problematic as their high computation time could result in non-optimized solutions.

In [11] however, POMDP was solved online (at small-time intervals) in a dynamic replanning scheme in belief space for the *Simultaneous Localization and Planning* problem (SLAP). They accomplished this by using the *Feedback-based Information Roadmap* (FIRM) method (a multi-query technique that reduces intractable dynamic programming to tractable dynamic programming) [17]. Additionally, their path planning system is not limited to a horizon, does not get confined to a local minima, and does not assume deterministic future observations [11]. One disadvantage of this method is that it assumes the environment (including the landmarks) are known to the robot, which makes its use in unknown environments difficult to evaluate.

B. Project Objective

The main objective of this project is to address the problem of simultaneous planning, localization, and mapping for safer robot path planning in uncertain and unknown environments. Solving this problem is essential to fully realize autonomous path planning. To be viable in real-world applications, our path planner will account for the following assumptions:

- The robot's location in relation to the surrounding map and obstacles is uncertain.
- The environment is unexplored *a priori* and on-board or external sensor measurements have uncertainty.
- Actions that lead to some goal state can only be considered 'optimized' if these actions are dependent on the results of localization (uncertainty of the robot's location must be within acceptable bounds).
- Calculating optimized states must be performed online and in small time intervals.

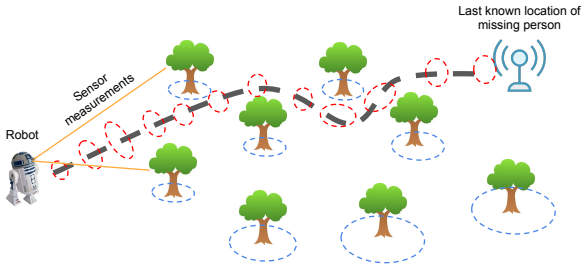

Fig. 1. **SCRAM Scenario.** The objective of our SCRAM method is to combine MPC, SLAM, and neural networks for path planning under uncertainty from both robot and obstacle locations. Using SCRAM, we enable the robot to safely reach some goal position while ensuring that the uncertainty bounds of the robot and obstacles never collide (preventing fatal collisions). This path planner can be useful in a search and rescue application, where the last location of a missing person can be set as the goal position.

- The dynamics of the robot must be satisfied. In this study we assume a point mass, however, it is typically assumed that even in dynamically complex robots, we can use discretized equations (see [18] and [19] for examples).

We aim to account for these assumptions by exploring three different avenues of research, namely MPC, SLAM, and neural networks. We combine these tools for an end-to-end framework that balances speed, safety, and uncertainty. More specifically, in our SLAM and MPC loop the SLAM module feeds the MPC module a state estimate. We improve our estimate by including state uncertainty via a predictive model of the state's uncertainty through neural networks for the MPC's finite horizon approximation and name these combined methods "SCRAM" (Fig. 1). Overall, we make two main contributions in this study: $(\mathbf{1})$ we demonstrate a learning approach to propagating state uncertainty for our MPC and SLAM combination, and $(\mathbf{2})$ we evaluate the feasibility of an online path planning framework using MPC, SLAM, and neural networks that accounts for uncertainty and map changes.

II. Methods

Our path planning architecture is built around an MPC framework, which is modified by SLAM for state estimation/localization, and neural networks for uncertainty propagation. Our methods are split into four sections: (1) we provide an overview of MPC and why MPC needs to be modified to account for uncertainty; (2) we explain our SLAM method and algorithm; (3) we describe further in detail our formulation of the neural network equations; and (4) we demonstrate how the overall SCRAM architecture combines MPC and SLAM with neural networks.

A. Model Predictive Control (MPC)

MPC can generate path planning online and continually satisfy the dynamic state of the robot [20] [21]. This is achieved by a control strategy that computes optimal control values and corresponding state values online, predict future outputs

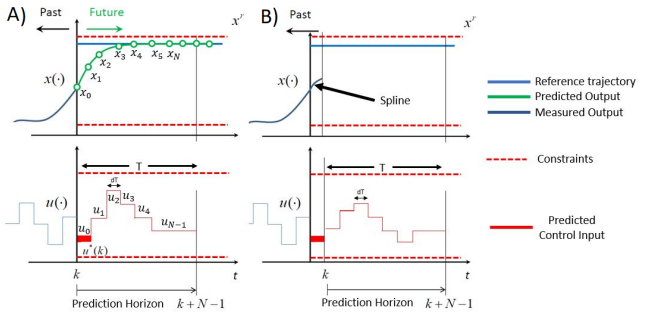

Fig. 2. **General Concept of Model Predictive Control (MPC).** The MPC aims to predict the future states of an agent within a finite time horizon in order to determine a next-best-action for the system [24].

over a time horizon, and minimize a objective function under constraints on both state and control variables (Fig. 2) [22]. Thus, MPC computes control actions through a deterministic representation of uncertainty, where deviation of the actual uncertainty produces a 'forecast error' that is counteracted by adjusting the control action of the next time step [23]. The goal of MPC is then to calculate a series of *control values* (e.g., in our case linear and angular velocity) that can manipulate the input or *state* variable (e.g., position x, y, and heading, θ) to a reference state. Figure 1 illustrates the concept of MPC, where x is the predicted output, and u is the control variable. The current sampling instant is shown by iteration k, where MPC calculates a set of N control values $[u(k+i-1), i = 0, 1, 2, 3, ...N-1]$ over a time interval T - this interval is also called the *prediction horizon* (equal to $N * dt$). The predicted outputs (calculated from control variables u), represent a set of x values optimized to reach the reference states $[x(k+i), k = 0, 1, 2, 3, 4, ...N)]$. The optimization of x values based on the control u is made by the *objective function* [20]. The objective function minimizes the sum of square predicted error outputs (in our case, the difference between the predicted state and reference state) as well as the square of control values while subject to constraints. All optimization calculations were done using python and CasADi modules [24].

For the reasons outlined above, MPC is a solid candidate for path planning in dynamic environments. However, due to the deterministic nature of traditional MPC, it is still inadequate when dealing with large uncertainty [25]. Although POMDPs are generally employed to address uncertainty, their solutions are typically intractable in small state spaces [10], they fail to generate adequate motion strategies when the environment is unknown [26], and arguably, more evidence is needed to show that POMDPs can be solved in reasonably short time steps [14]. Yet, we still require a stochastic approach to our active SLAM problem as the actual position of the robot can differ greatly from the predicted position. Also, the robot cannot know beforehand which measurements will be acquired in the future (i.e., which new landmarks will be observed that may lead to better localization). Thus, in this study, we will use neural networks to predict how uncertainty is propagated in the MPC prediction horizon for more optimal trajectory planning.

Algorithm 1: FastSLAM 2.0

Initialize $x^i_{0|0}, i = 1, ..., N$;
Initialize $\alpha^i_{0|0} = \frac{1}{N}, i = 1, ..., N$;
for $k = 1 : T$ **do**
  $z_k \leftarrow$ *Observation*(u_k);
  $x^i_{k+1|k} \leftarrow$ *PredictionStep*$(x^i_{k|k}, z_k)$;
  $x^i_{k+1|k+1}, \alpha^i_{k+1|k+1}, m^i_{k+1} \leftarrow$ *UpdateStep*$(x^i_{k+1|k}, \alpha^i_{k|k}, z_k, m^i_k)$;
  $x^*_{k+1|k+1} \leftarrow$ *ChooseBestParticle*$(x^i_{k+1|k+1}, \alpha^i_{k+1|k+1})$;
  if $N_{eff} \leq N_{thresh}$ **then**
    $x^i_{k+1|k+1}, \alpha^i_{k+1|k+1} \leftarrow$ *ResampleParticles*();
  end
  return $x^*_{k+1|k+1}$
end

B. Simultaneous Localization and Mapping (SLAM)

In most cases, implementations of model predictive control assume perfectly known states; however, this is not a valid assumption within real-life scenarios. Thus, we use simultaneous localization and mapping (SLAM) to estimate our robot's state. Furthermore, as the map of the environment is not known *a priori*, we can tap into SLAM's mapping capabilities to provide the MPC with an up-to-date map in its collision-checking optimization. In this project, we used the Rao-Blackwellized particle filter-based FastSLAM 2.0 [27] with 250 particles and stratified resampling to capture the true underlying state probability distribution. While the exact formulation is out of the scope of this paper, a high-level overview of the algorithm is described in Algorithm 1. In this project, we used a heavily customized version of the PythonRobotics [28] FastSLAM2 implementation for the SLAM module. We used a max observation range of 6 meters, state transition and observation noise covariances of

$$Q = \begin{bmatrix} 5.0 & 0 \\ 0 & 0.35 \end{bmatrix}, R = \begin{bmatrix} 0.6 & 0 \\ 0 & 0.17 \end{bmatrix}$$

and simulated state transition and observation noise covariances of

$$Q_{sim} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.017 \end{bmatrix}, R_{sim} = \begin{bmatrix} 0.75 & 0 \\ 0 & 0.17 \end{bmatrix}$$

and a yaw noise rate of 0.01.

C. Neural Networks

As we are using a deep neural network to predict uncertainty propagation, we have a training stage and a testing stage (Fig. 5 and Fig. 6). The deep neural network was built with three hidden layers (Fig. 3). The first layer consists of 53 units. The first three units correspond to the x, y and heading of the robot. The next 30 inputs consist of the x and y positions of

each landmark as well as the distance from the robot to the landmark. For ten landmarks this makes a total of 30 input values. We then append to the input array how many times in each second we have seen the landmark. We do this for all 10 landmarks which adds another 10 inputs to the input array. The last 10 inputs consist of a one hot encoding array of whether the landmark is visible or not from our current position. This bring the total of inputs to 53.

The first hidden layer consists of 512 neurons, followed by a layer consisting of 256 neurons and then a final hidden layer containing 64 neurons (Fig. 3). Each one of these layers are fully connected and use Rectified Linear Units (ReLU) [29] as activation functions. The final output layer consisting of 4 outputs uses a Linear activation function. The four output neurons correspond to the the four values of the robot's 2×2 covariance matrix. We have chosen to use a Linear activation function since the outputs can take any value and are not restricted to values between 0 and 1 as is usually the case with deep neural networks. Hence we have not chosen typical output layer activation functions such as softmax or hyperbolic tangent.

During training, we use Mean Squared Error (MSE) as our loss function. The mathematical formula for MSE is:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y - \hat{y})^2$$

Here N is the total number of instances, y is the ground truth and $\hat{y}$ is the prediction. The MSE is a commonly used loss function when we are trying to optimize the neural network for regression problems. Since our goal is to approximate the covariance matrix, which contains real valued entries, it is appropriate that we choose MSE as our loss function as opposed to Log Likelihood. We use a total of 30 epochs for training on the full dataset, which consists of 100 randomly generated maps, each containing 10 landmarks. Prior to training we also shuffle and scale the dataset. We save the parameters we obtain upon scaling so that we can use them to pre process the data during testing as well.

D. Combining MPC, SLAM, and Neural Networks

Fig. 4 demonstrates our overall SCRAM architecture. First, we have our motion model ((1)):

$$X_{k+1} = A * X_k + B * U_k + C * w_k \tag{1}$$

where $\mathbf{X} = \begin{bmatrix} x, & y, & \theta \end{bmatrix}^\intercal$ represents our state variables (position and heading), and $\mathbf{U} = \begin{bmatrix} v_x, v_y, \omega \end{bmatrix}^\intercal$ represents our control variables (velocity and angular velocity). We also add a random Gaussian noise for our control variables at each time step ($w_k \sim \mathcal{N}(\mu, N)$). Matrices A, and B represent the dynamic equations used for our model. In this case, we assume a simple point mass and kinematic equations (however, these matrices can easily be modified to incorporate more complex dynamics). Vector C represents the standard deviations (for velocity and angular velocity) which are multiplied by w_k (the noise term).

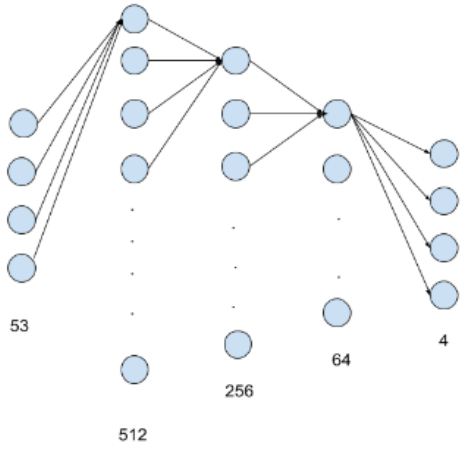

Fig. 3. **Neural Network Architecture** Deep neural network with three hidden layers. The hidden layers have a ReLU activation function while the output layer has a Linear activation function. The network was trained for 30 epochs.

Next we have the objective function (e.g., equation (2)):

$$\min_{U_k,\ldots,J_{k+N}} \sum_{k=0}^{N-1} Q * \left(X_k^{curr} - X_k^{ref}\right)^2 + R * U_k^2 \tag{2}$$

The goal of our objective function is to find the optimal control value that (while accounting for a future number of states determined by N) effectively minimizes the distance between the current state (X_k^{curr} ; retrieved from SLAM) and the reference state (X_k^{ref}). The gain matrix Q was selected to be an identity matrix, as we want to put equal cost on all of our states. For gain matrix R, we choose 0.5 for velocity, and 0.05 for angular velocity. The reason for this choice is to place a greater cost for controlling the angular velocity (doing so ensures more smooth rotations of the robot).

The control values calculated by the objective function are also under two type of constraints: hard constraints and changing constraints. We omitted constraining our state values (thus, minimum and maximum values ranged from negative to positive infinity). However, we constrained our velocity and angular velocity ($V_k^{\min} = -1.5m/s$ (in x and y directions), $V_k^{\max} = 1.5m/s$ (in x and y directions), $\omega_k^{\min} = -\pi/2$, $\omega_k^{\max} = \pi/2$). These values were chosen empirically (i.e., graphically determined which values provided reasonable motions).

We also have a set of equality constraints ((3):

$$X_{k+1} - f\left(X_k, U_k\right) = 0 \tag{3}$$

These equality constraints are part of the multiple shooting optimization method ([30]). These constraints ensure that the control values chosen by the optimization function obey the discretized equations of the motion model (i.e., the left hand side of the motion model must equal the right hand side of the motion model for all time steps).

Lastly, we have changing inequality constraints:

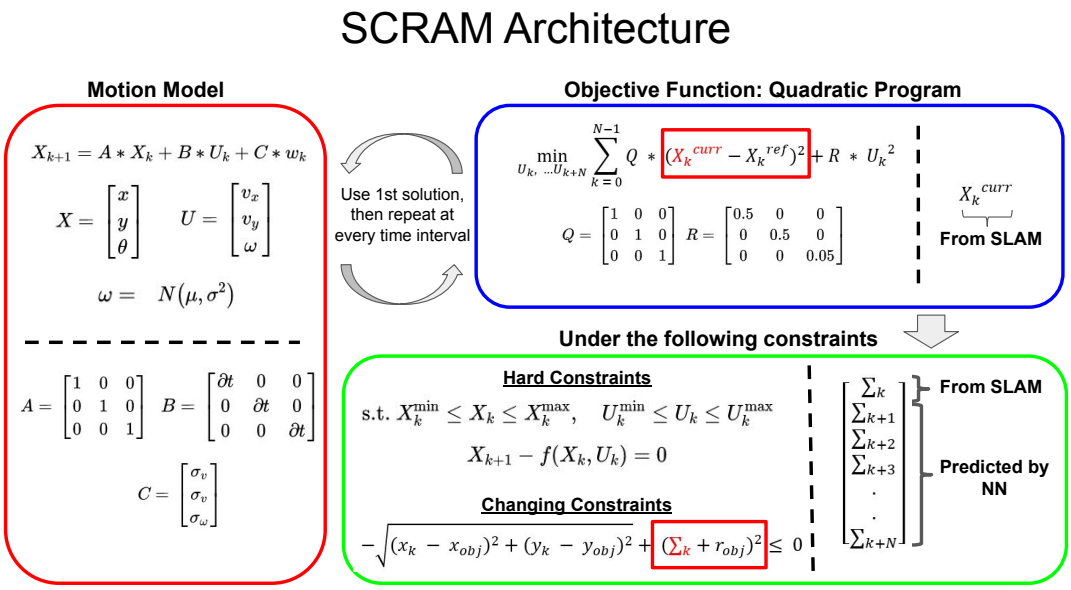

Fig. 4. **SCRAM Architecture.** To summarize, the motion model is assumed to be a point mass with additive Gaussian noise. The objective function finds a optimal control value that minimizes the difference between the current and reference state at all time steps within the MPC prediction horizon. Lastly, the objective function is under both hard and changing constraints. The hard constraints force the computed control solutions to be within a reasonable range (we do not put hard constraints on our state values so that the robot can freely move to any point in the map). The changing constraints ensure that the uncertainty boundary of the robot and surrounding obstacles do not collide within the MPC prediction horizon.

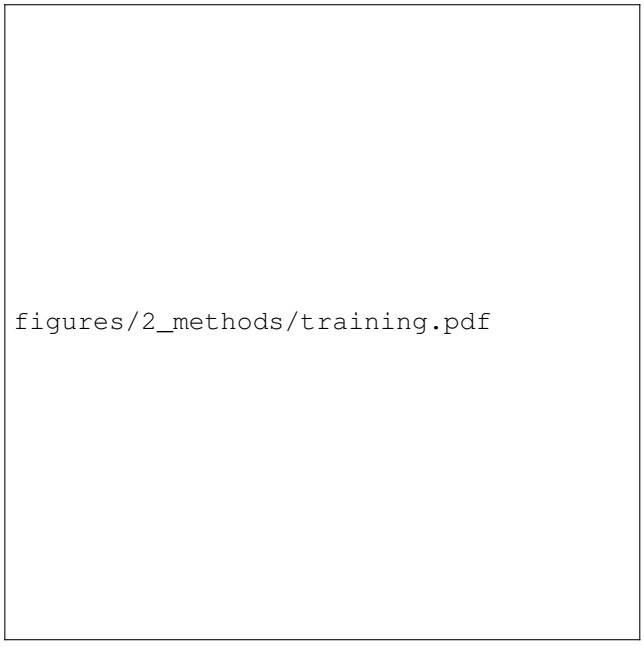

Fig. 5. **SCRAM Training.** This figure demonstrates the training procedure of our SLAM method. First we select N different maps, where obstacles in each map are randomly distributed. A simulation is run on this map, where the robot goes from an initial to goal position. At each time step, an observation is taken, Z_k^{curr} (i.e., we obtain the current sensor measurements and map features). These measurements are used as the input to particle-filter SLAM, which produces two outputs utilized in MPC (X_k^{curr}), and (Σ_k^{curr}), producing output U_k^{mpc}. For every map at every time step, the current measurements/map features, state position and uncertainty (along with other variables; see neural network section in methods) are entered into a large database. Finally, our neural network is trained on this database to produce our neural network model.

Fig. 6. **SCRAM Testing.** In the testing procedure (where we run our simulation on a map the robot has not seen before), the overall process is similar to the training procedure. The main difference is that the output of our particle-filter SLAM (X_k^{curr}) and the simulated observation (Z_k^{curr}) are used as part of the inputs to the neural network model (along with the other variables; see neural network section in the methods). While the current state uncertainty (Σ_k^{curr}) can be retrieved from SLAM, Σ_k to Σ_{k+N} is predicted by our neural network model.

$$-\sqrt{(x_k - x_{obj})^2 + (y_k - y_{obj})^2} + (\Sigma_k + r_{obj})^2 \leq 0 \quad (4)$$

These constraints ensure that the uncertainty boundary of the robot never collides with the uncertainty boundary of all surrounding obstacles (which are detected by the robot at a user-specified range). x_{obj} and y_{obj} represent the x and y position (or center) of all obstacles detected by the robot (we assume the positions of the obstacles do not change from the current time step in the prediction horizon). x_k and y_k represent the x and y position of the robot for time step k to time step $k+N$. Σ_k represents the radius of the uncertainty boundary of the robot, and r_{obj} represents the radius of the uncertainty boundary of the obstacle. Note that at time step k, Σ is estimated by SLAM, while from time step $k+1$ to time step $k+N$, Σ is predicted by our neural network formulation. Thus, our objective function considers the propagation of robot uncertainty along its prediction horizon, and chooses control values that ensure uncertainty of the robot and obstacles do not collide along its path.

III. Results

In this section we describe the results of our neural network implementation and our SCRAM framework in a simulation environment. More specifically, we provide analyses to show that our network is reasonable during both the training and testing procedures, and we show that inferring future covariances for the model predictive controller enabled our simulated robot to faster traverse a cluttered environment (which is assumed to be initially unknown and unexplored).

A. Neural Network

We performed two different tests for our neural network. In the first experiment, we consolidated all 101 maps and performed a 5-fold cross validation on them. The second test was performed using 100 maps as the training data and 1 map as testing. We will discuss both experiments.

The first experiment was done to ensure that the prediction of the covariance matrix was reasonable across multiple maps and to test if the model could learn the parameters from different combinations of maps. During the test phase of this experiment, we did not pay attention to the order of the testing instances. We were only concerned with the accuracy of our predictions. Fig. 7 shows in red the error of the first cross validation fold. The error is defined as the absolute value difference between the predicted value and ground truth. The black line represents the standard deviation. We observe that for the 5-fold cross validation, the standard deviation was 0.12.

For the second experiment, the testing instances were fed to the neural network in the order that they were generated (Fig. 8). There were a total of 127 instances, meaning that the robot took 127 steps to reach the goal state. We observe that initially the covariance matrix error is high and then drops as the robot continues traversing the map. We believe that the reason behind this is due to the fact that when the robot starts off in its trajectory, there is a high degree of uncertainty

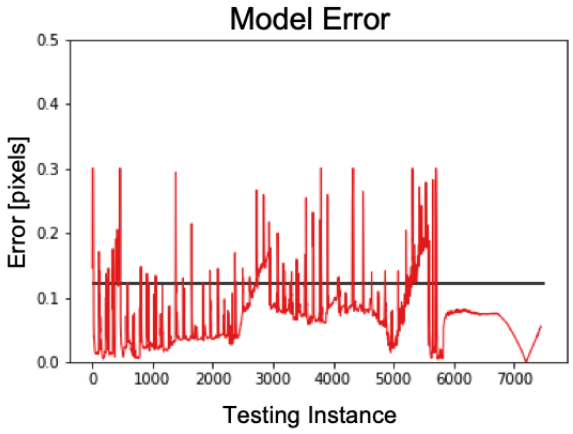

Fig. 7. **NN Experiment 1: 100 Maps, 5-Fold.** Average error of the neural network model in 5-fold cross validation testing. The train/test (80/20) splits combined consisted of 100 maps.

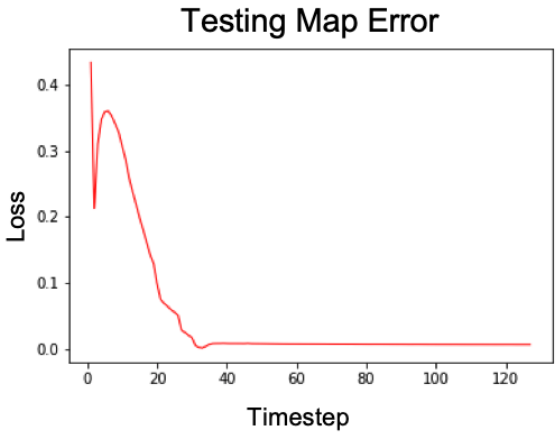

Fig. 8. **NN Experiment 2: Single Map Prediction.** The neural network was trained on 100 maps and tested on a single map which it had not encountered previously. The error is high in the first few time steps but reduces in future time steps. By the time the robot reaches the 40th time step, the Network is able to accurately predict the covariances.

and consequently, it is hard for the neural network to predict this uncertainty. As the robot progresses, the uncertainty also reduces and as a result, it becomes easier for the neural network to predict the values. We also show the graph of training loss (Fig. 9). As we can see, at the first epoch, the loss is at around 0.008, but as we keep training, the loss reduces to around 0.0016. This reduction in loss is a strong indicator that the neural network is able to learn some mapping between the inputs and the outputs.

B. Simulation Experiments

1) Setup: Our simulation setup consisted of ten randomly placed landmarks within a 10x10 meter grid, with the robot's objective to move from the bottom left corner $[-10, -10]$ to the top right corner $[+10, +10]$ (Fig. 10). For SLAM, we used 250 particles chosen empirically with an observation range of

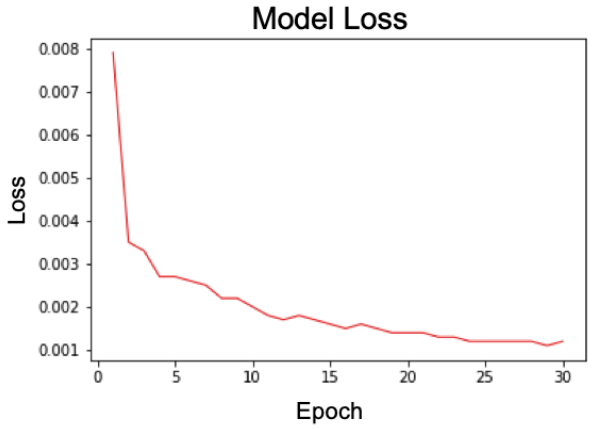

Fig. 9. **Model Loss.** The neural network was trained over 30 epochs. Initially we have a loss of around 0.008. By the time the training procedure is complete, our loss shrinks to 0.0016. This result is a good indicator that our network is learning the relationship between the inputs and the outputs.

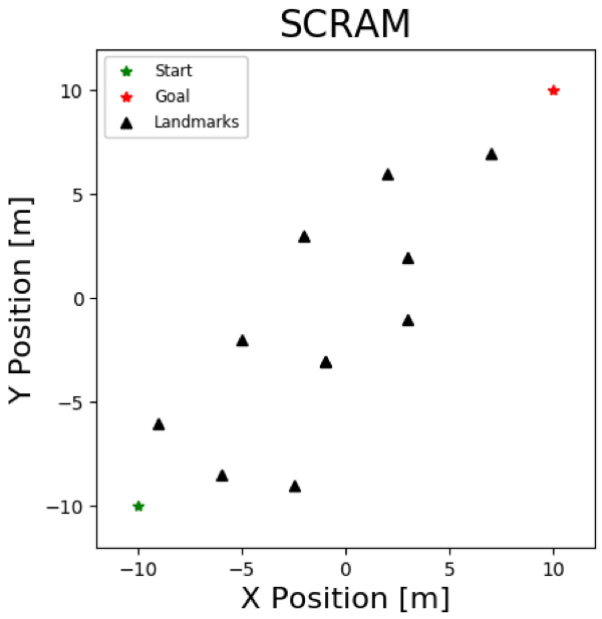

Fig. 10. **Simulation Setup.** Our experimental setup consisted of ten randomly placed landmarks within a 10x10 meter grid. The objective of our robot was to move from the bottom left corner $[-10, -10]$ to the top right corner $[+10, +10]$.

6 meters. For MPC, we used a finite time horizon of $N = 20$ time steps and a 5 meter observation range. Finally, prior to testing we trained a neural network model on 100 randomly-generated maps using the architecture described in Section II. Note that the testing map described in this section was held out from the training set.

2) Results: We compared our methods against a naive approach that did not incorporate covariance predictions via neural networks, i.e. a framework that used only SLAM and MPC modules. In the naive approach, we observed that our agent prioritized safety over speed and hesitated when

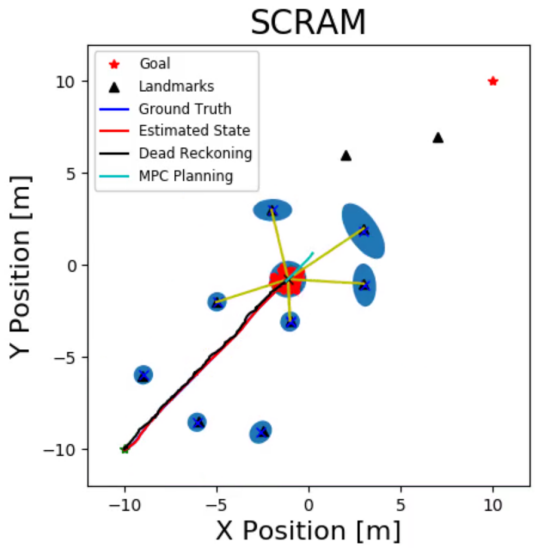

Fig. 11. **Simulation Example.** An example time step during the simulation process using neural networks to predict future covariances for the MPC module. At each time step, the neural network provides the MPC with covariances of each finite time horizon state to determine a more optimal control.

passing through tight spots in order to ensure no collisions. Furthermore, it performed wide turns to guarantee landmark avoidance. In this map, the robot took approximately 20 seconds (at 15 frames per second) [1]. In comparison, using our trained neural network model to predict covariances, we observed much more aggressive behavior in our agent that prioritized speed over safety. In particular, our robot passed through tighter spots with greater ease and less hesitation, and makes sharper turns around landmarks. This behavior resulted in a trajectory that was four seconds faster than the naive method [2]. Fig. 11 provides graphical intuition of our simulation setup, with blue robot/landmark covariances overlaid on top of the ten randomly placed landmarks. Aligned with our observations of the robot's behavior, the light-blue MPC path planning line provided our agent with much more aggressive control inputs when taking future state covariances into consideration.

Computationally, the addition of our neural network module was negligible and did not provide a noticeable slowdown to the system. As we trained our neural network model offline, inference with our model at each time step showed no additional computational expense, and optimization within the MPC was just as fast as the naive approach. Therefore, we argue that video speed is a reasonable metric for computational expense between the two frameworks and conclude that the speed increase by including covariance prediction outweighs any potential slowdowns with the addition of the network.

[1] https://youtu.be/4bDE0C2takw
[2] https://youtu.be/iafgmHSLPw8

IV. Discussion

In this paper, we present a novel path planning architecture that incorporates SLAM, MPC, and neural network components. More specifically, we trained a neural network model to predict state uncertainty in the finite horizon of an MPC and demonstrated more aggressive, faster behavior while still maintaining a safe distance between the uncertainty boundaries of the robot and the landmarks. To maintain realism, we used a particle filter-based SLAM algorithm for state estimation, uncertainty prediction, and map building at each time step. Our MPC took these inputs to generate a finite horizon state trajectory, then inferred future covariances using our neural network model for its optimization statement.

From these results and our graphs, we can see that the neural network is indeed able to capture the relationship between our chosen inputs and the robot's covariance at each provided time step. However, we recognize that several variables could have impacted our results, namely the gain matrices in MPC, our chosen SLAM state and observation noise covariances, and our chosen input variables for the neural network model. These 'tuning' parameters can always be further optimized to demonstrate potentially improved results. For future work, we plan to improve our network by increasing its size and training it on more data that may better capture the functional relationship between our inputs and outputs. We also need to further validate our methods by comparing our learning approach to uncertainty prediction with more implicit formulas (e.g., Bayesian probabilities) utilized in stochastic model predictive control (SMPC) [25]

In addition, we plan to use recurrent neural networks (RNNs) to predict the finite time horizon covariances. RNNs have been shown to better handle time series data ([31], [32], [33], [34]) and would be a reasonable choice as there is a temporal factor when predicting robot and landmark covariances. This problem could also be formulated within a reinforcement learning (RL) framework in that the robot learns noise covariances and perhaps a global trajectory through trial-and-error to maximize some provided notion of reward.

Lastly, because MPC works well for dynamically complex robots, our objective is to scale our methods from a point mass assumption to a quadruped robot (Fig. 12). Using Gazebo software (Fig. 13), we can demonstrate our methods in a 3-dimensional environment, and then on the actual hardware. This is all towards a framework for autonomous agents that can further increase its trajectory speed while maintaining its personal safety.

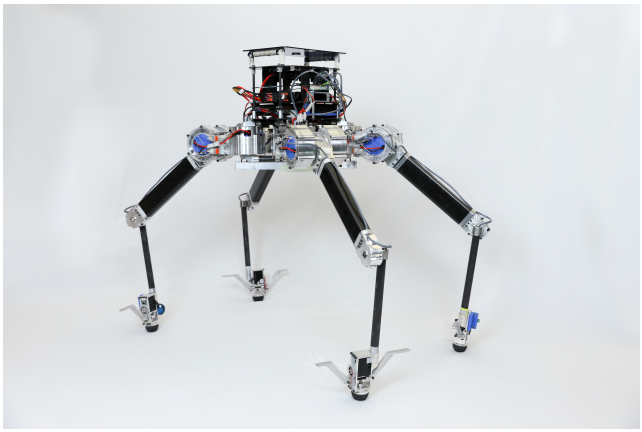

Fig. 12. **Quadruped.** An example of a quadruped robot that we can use to test our methods, called ALPHRED (Autonomous Legged Personal Helper Robot with Enhanced Dynamics) [35].

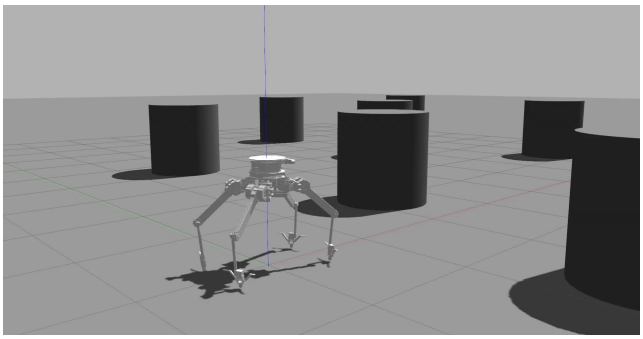

Fig. 13. **Quadruped in Gazebo.** ALPHRED robot shown in a Gazebo simulation.

Acknowledgments

We would like to thank Dr. Ankur Mehta and Alexie Pogue for their instruction this quarter and for this opportunity.

References

[1] D. Scaradozzi, S. Zingaretti, and A. Ferrari, "Simultaneous localization and mapping (slam) robotics techniques: a possible application in surgery," *Shanghai Chest*, vol. 2, no. 1, 2018. [Online]. Available: http://shc.amegroups.com/article/view/4083

[2] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, 1959. [Online]. Available: https://doi.org/10.1007/BF01386390

[3] P. Hart, N. Nillson, and B. Raphaell, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, 1968. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4082128

[4] S. M. Chaves, "Belief-space planning for active visual slam in underwater environments." 2016.

[5] L. E. Kavraki, P. Svestka, J. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, Aug 1996.

[6] S. LaValle and J. Kuffner, "Randomized Kinodynamic Planning," *The International Journal of Robotics Research*, 2001. [Online]. Available: https://doi.org/10.1177/02783640122067453

[7] P. Sudhakara, V. Ganapathy, and K. Sundaran, "Trajectory planning using enhanced probabilistic roadmaps for pliable needle robotic surgery," pp. 61–64, March 2018.

[8] L. Chen, Y. Shan, W. Tian, B. Li, and D. Cao, "A fast and efficient double-tree rrt*-like sampling-based planner applying on mobile robotic systems," *IEEE/ASME Transactions on Mechatronics*, vol. 23, no. 6, pp. 2568–2578, Dec 2018.

[9] A. Sieverling, N. Kuhnen, and O. Brock, "Sensor-based, task-constrained motion generation under uncertainty," pp. 4348–4355, May 2014.

[10] P. Kaelbling, M. Littman, and A. Cassandra, "Planning and acting in partially observable stochastic domains," *Elsevier*, 1995. [Online]. Available: https://doi.org/10.1016/S0004-3702(98)00023-X

[11] A.Aghar-Mohammadi, S. Agarwal, S. Chakravorty, and N. Amato, "Simultaneous localization and planning for physical mobile robots via enabling dynamic replanning in belief space," *CoRR*, vol.

abs/1510.07380, 2015. [Online]. Available: http://arxiv.org/abs/1510.07380

[12] T. Smith and R. Simmons, "Point-based pomdp algorithms: Improved analysis and implementation," pp. 542–549, 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=3020336.3020402

[13] D. Grady, M. Moll, and L. E. Kavraki, "Automated model approximation for robotic navigation with pomdps," pp. 78–84, May 2013.

[14] W. V. der Hof, "Robot search in unknown environments using POMDPs," *TU Delft University Thesis*, 2014. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid:5902cc25-6d45-4ea2-9a8a-4098874e8444?collection=education

[15] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE Robotics Automation Magazine*, vol. 13, no. 2, pp. 99–110, June 2006.

[16] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (slam): part ii," *IEEE Robotics Automation Magazine*, vol. 13, no. 3, pp. 108–117, Sep. 2006.

[17] A. Agha-mohammadi, S. Chakravorty, and N. Amato, "FIRM: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements," *The International Journal of Robotics Research*, 2013. [Online]. Available: https://doi.org/10.1177/0278364913501564 8

[18] J. Hooks and D. Hong, "Implementation of a versatile 3d zmp trajectory optimization algorithm on a multi-modal legged robotic platform," pp. 3777–3782, Oct 2018.

[19] A. W. Winkler, F. Farshidian, D. Pardo, M. Neunert, and J. Buchli, "Fast trajectory optimization for legged robots using vertex-based zmp constraints," *IEEE Robotics and Automation Letters*, vol. 2, pp. 2201–2208, 2017.

[20] A. H. T. H. T.H Mohamed, H. Bevrani, "Decentralized model predictive based load frequency control in an interconnected power system," *Elsevier, Energy Conversion and Management*, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0196890410004188

[21] Marwaha, Lather, Dhillon, "Multi area load frequency control using model predictive controller," *Proceedings of International Conference on Electrical and Electronics Engineering*, 2015. [Online]. Available: https://www.researchgate.net/publication/290439146_MULTI_AREA_LOAD_FREQUENCY_CONTROL_USING_MODEL_PREDICTIVE_CONTROLLER

[22] S. Xin, R. Orsolino, and N. Tsagarakis, "Online relative footstep optimization for legged robots dynamic walking using discrete-time model predictive control," 03 2019.

[23] C. Leung, S. Huang, and G. Dissanayake, "Active slam using model predictive control and attractor based exploration," pp. 5026 – 5031, 11 2006.

[24] Mehrez Said, "Optimization based solutions for control and state estimation in non-holonomic mobile robots: Stability, distributed control, and relative localization," *Cornell University, arXiv.org*, 2018. [Online]. Available: https://arxiv.org/abs/1803.06928

[25] A. Mesbah, "Stochastic model predictive control: An overview and perspectives for future research," *IEEE Control Systems Magazine*, vol. 36, no. 6, pp. 30–44, Dec 2016.

[26] H. Kurniawati, T. Bandyopadhyay, and N. Patrikalakis, "Global motion planning under uncertain motion, sensing, and environment map," *Autonomous Robots*, vol. 33, pp. 255–272, 2011.

[27] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit *et al.*, "Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges."

[28] A. Sakai, D. Ingram, J. Dinius, K. Chawla, A. Raffin, and A. Paques, "Pythonrobotics: a python code collection of robotics algorithms," *arXiv preprint arXiv:1808.10703*, 2018.

[29] Y. Bengio, X. Glorot, and A. Bordes, "Deep sparse rectifier neural networks," 2011. [Online]. Available: http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf

[30] M. Kiehl, "Parallel multiple shooting for the solution of initial value problems," *Elsevier, Parallel Computing*, 1994. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016781910680013X?via3Dihub

[31] G. Dorffner, "Neural networks for time series processing," *Neural Network World*, vol. 6, pp. 447–468, 1996.

[32] J. Yuan, H. Wang, C. Lin, D. Liu, and D. Yu, "A novel gru-rnn network model for dynamic path planning of mobile robot," *IEEE Access*, vol. 7, pp. 15 140–15 151, 2019.

[33] M. Bency, A. Qureshi, and M. Yip, "Neural path planning: Fixed time, near-optimal path generation via oracle imitation," *CoRR*, vol. abs/1904.11102, 2019. [Online]. Available: http://arxiv.org/abs/1904.11102

[34] B. Ni, X. Chen, L. Zhang, and W. Xiao, "Recurrent neural network for robot path planning," *Parallel and Distributed Computing: Applications and Technologies*, vol. 3320, 2005. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-540-30501-9_43

[35] Hong, "Romela — robots, romela.org," *RoMeLa*, Accessed: 6-Sep-2019. [Online]. Available: http://www.romela.org/robots/