Flow Control of Wireless Mesh Networks Using LQR and Factor Graphs

Ryan Darnley

CMU-RI-TR-21-58

August 2021

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Thesis Committee:
Dr. Matthew Travers, Chair
Dr. Anthony Rowe
John Miller

Submitted in partial fulfillment of the requirements for the degree of Master's of Science in Robotics.

Abstract

Advances in the estimation and more recently the controls communities have leveraged the use of factor graphs to overcome problems of scalability and runtime in dynamical systems. In continuation of that work, we look to use factor graphs to perform LQR control on a wireless mesh network (WMN). Inspired by the communication challenges presented in the DARPA Subterranean Challenge, we look to model a WMN as a dynamical system capable of control. Furthermore, we present WMNLQR, a network flow control algorithm using LQR and factor graphs that is capable of achieving linear runtime growth with respect to both trajectory length and state dimension. To illustrate the real time capabilities of our algorithm, we conduct a thorough analysis of WMNLQR versus four other LQR solvers. We find that even more advanced algorithms utilizing dynamic programming can become intractable for real time centralized control as the state space increasingly grows.

Furthermore, we present a novel decentralized control algorithm, DWMNLQR. This algorithm leverages the physical RF links between communication nodes to assist the message passing process used to solve a factor graph. By exploiting the sparsity of factor graphs, we find that a globally optimum solution can still be achieved despite framing the communication network as a series of connected local subproblems. Specifically, we iteratively compute the LQR solution to each communication node individually, with each node only aware of its local mesh. By eliminating the need to communicate to a centralized control node, DWMNLQR has improved both system robustness and runtime for real world scenarios.

Acknowledgements

I would first like to thank the members of my committee: Dr. Matthew Travers, Dr. Anthony Rowe, and John Miller. Without their direction and insight, I would not have learned nearly as much as I did in producing this report.

I would also like to thank all members of CMU's SubT Challenge team, Team Explorer - Sebastian Scherer, Steve Willits, David Kohanbash, and many more. The challenge has been a demanding but very rewarding experience and I have had the great privilege of becoming friends with many great engineers. I very much look forward to the final competition this coming September.

Finally I would like to thank my mom, dad, and two sisters for always being there for me.

Contents

List of Figures

Chapter 1

Background

1.1 The DARPA Subterranean Challenge

1.1.1 What is the SubT Challenge?

The Defense Advanced Research Projects Agency (DARPA) Subterranean (SubT) Challenge is a multi-year competition set forth by DARPA to augment the capabilities of war fighters and search and rescue operators in underground environments. The competition consists of teams spanning the globe, all creating autonomous robotic platforms designed to traverse and map complex and unknown underground environments. Fundamentally, the objective of the robot teams is to identify objects of interest known as artifacts. There are 10 different artifact types, including human survivors, backpacks, gas leaks, etc. After identifying and localizing these artifacts, the robotic platforms will attempt to relay the location of said artifacts back to human operators located at the basestation. From there, the operator can submit artifacts that have been deemed accurate and try to score 1 point for each successful submission.

Figure 1.1 further explains the details of the competition. Specifically, the teams deploy their systems in three different subterranean environments: tunnels, urban buildings, and caves. All three environments possess unique characteristics which impede the performance of the robot teams. Among the many challenges, the one which serves as chief inspiration to this report is the problem of communication networking. [3]

1.1.2 The Communication Network Problem

In an attempt to help the robots explore the unmapped terrain and report identified artifacts to the human operators, a communication network is deployed. The chief purpose of the communication network is two-fold. First and most important, the network provides the infrastructure needed to relay artifact detections from the robots to the basestation so that they can be analyzed and submitted. Second, the communication network provides human-in-the-loop manipulation and helps robots share odometry, maps, and other information with one another to optimize and coordinate their exploration strategy. Optimized traversal of the terrain subsequently

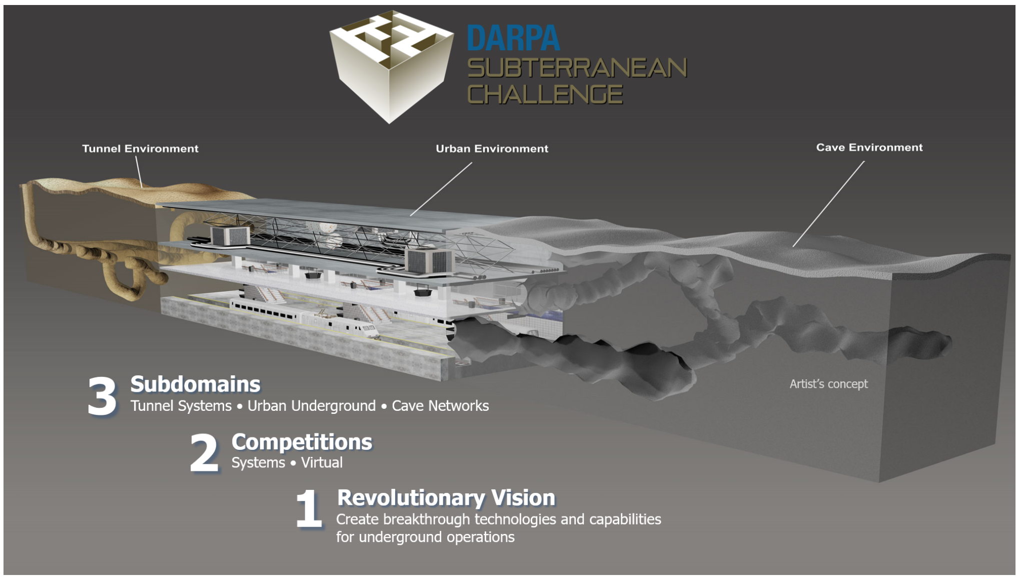

Figure 1.1: Visualization of the three different underground environments used in DARPA SubT Challenge [3]

improves the performance of the artifact detection and the team's overall score.

The three different underground environments presented by DARPA all possess unique characteristics, each of which creates a unique problem. Some of the many communication problems faced by the robots in the competition include non-line-of-sight conditions, multipath, dynamic levels of signal strength, porous walls and the like. In order to overcome these challenges, teams attempt to create ad-hoc communications networks that are both robust and capable of handling fluctuating rates in traffic. Overall, there are three main strategies used to create the communications network: wireless mesh networks (WMN), ethernet connections, and data mules.

1.1.3 CMU's Communication Network Strategy

In an attempt to solve the communication network problem, Team Explorer from Carnegie Mellon University, selected the wireless mesh network (WMN) strategy. As the foundation of the network, Team Explorer uses 5.0GHz Rajant DX2-50 radios as seen in Figure 1.2 [18]. For easier deployment, the radios have been removed from their blue outer casing and placed inside of a green fiberglass tube. These tubes, as seen in Figure 1.3, are then loaded into a device called the node dropper. The dropper, seen fully in 1.3, contains solenoid actuated locks which individually release the nodes when triggered.

While the robots explore the terrain, a ROS node called the comms-planner runs in the background. This node monitors each robot's current distance from other dropped communication nodes as well as the robot's line-of-sight conditions to other dropped communication nodes. Specific distance and line-of-sight conditions then trigger the robot to actuate one of the locks and release one of the communication nodes when appropriate.

Figure 1.2: Rajant DX2-50 radio [18]

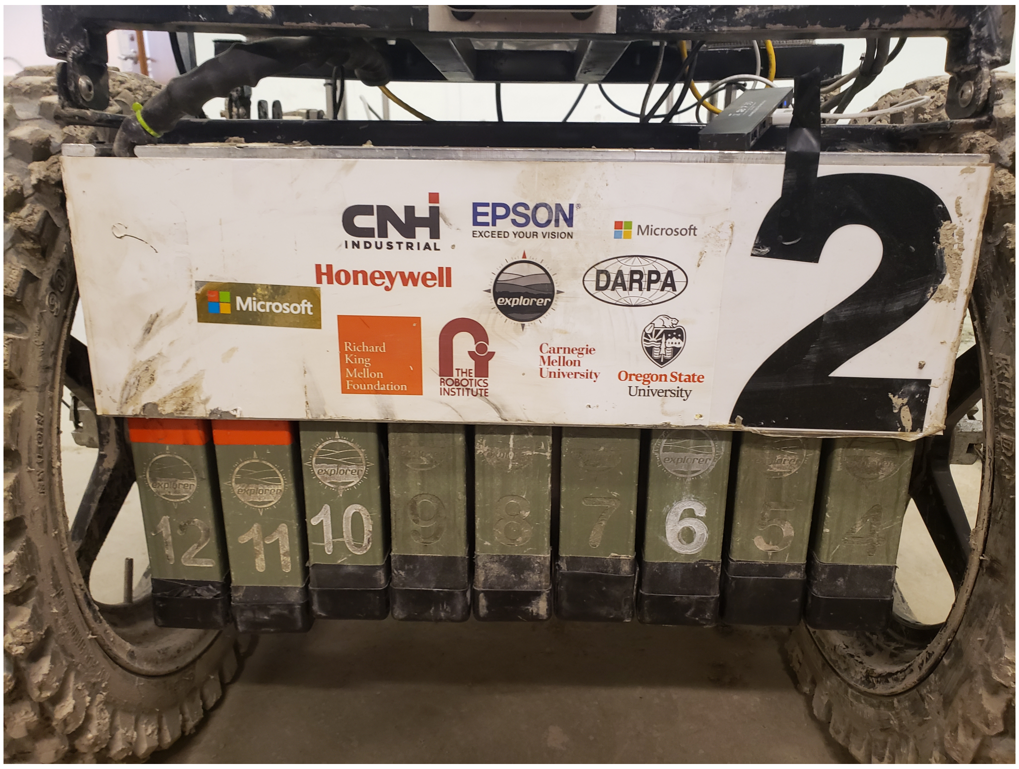

Figure 1.3: Node dropper loaded with 9 node casings each holding a Rajant DX2-50 radio

Ultimately, the ad-hoc wireless mesh network continually grows in size as the robots continue to explore away from the basestation.

A more clear representation of this phenomenon can be seen in Figure 1.4. The figure is a screenshot from an Rviz GUI showing the odometry (inside the blue ellipse) and pointcloud map from two ground vehicles - UGV2 and UGV3. The black dot inside each red circle is the location of a communication node dropped by one of the ground robots as it explored the terrain. This specific illustration is small in scale. During normal competition, up to 24 nodes are deployed from the three ground robots on the course. This number is then augmented by the total number of robots deployed on the course (up to 9) plus the basestation, each of which is fitted with a Rajant DX2-50 radio.

In addition to the ad-hoc deployment of the communication network, the traffic within the network is also very volatile. This volatility indicates the need to use flow control algorithms. Traditionally, there are many

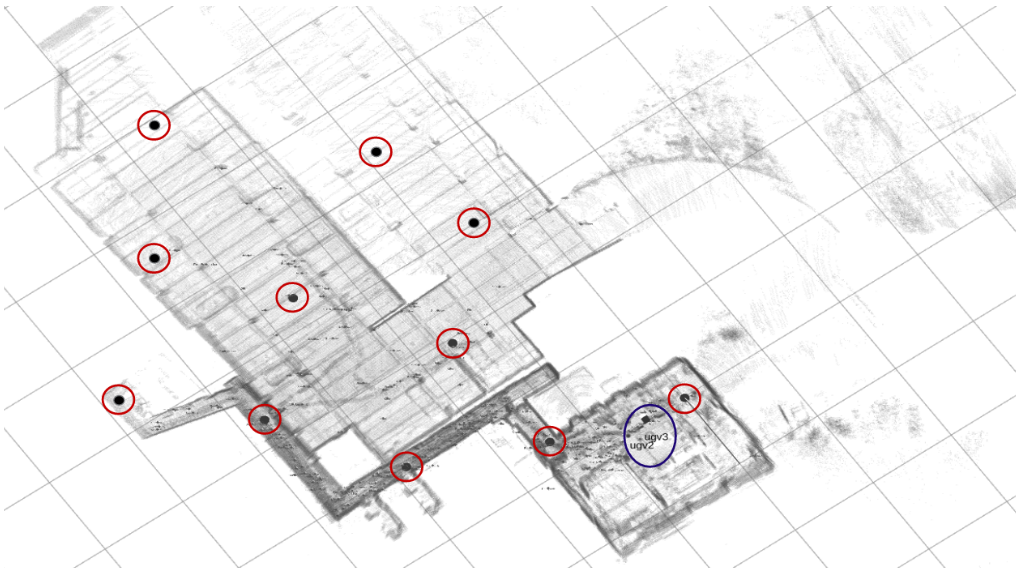

Figure 1.4: Screenshot of Rviz display showing Rajant nodes deployed during ground robot test run

heuristic flow control designs used in wireless networking. These include but are not limited to no flow control, token bucket, and fixed rate. Specifically the token bucket implementation is used by CMU's Team Explorer for the SubT Challenge. Given the dynamics of the environment, however, there is the opportunity to frame the network flow control as an optimal control problem (OCP). Prior to discussing the approach and results achieved in this paper, however, a background on modern control theory and specifically the LQR controller is presented.

1.2 Modern Control Theory

1.2.1 Linear Quadratic Regulator

The Linear Quadratic Regulator (LQR) is a full-state feedback controller. As per its name, it uses linear control laws to control linear dynamical systems while optimizing a quadratic cost function. Equation 1.1 shows the equation for a linear dynamics problem.

$$x_{t+1} = Ax_t + Bu_t \tag{1.1}$$

In this equation, the state space of the problem is represented by $x_t \in \mathbb{R}^{\ltimes}$ where t is the discrete time-step and n is the dimension of the state space. The control space is represented by $u_t \in \mathbb{R}^m$ where m is the dimension of the control space.

Connecting each timestep together are the state transition matrix (Jacobian), A, and the control matrix, B. The state transition matrix, $A \in \mathbb{R}^{n\times n}$, is a constant matrix that represents the temporal and spatial dependencies between elements of the state space. Meanwhile, the control matrix, $B \in \mathbb{R}^{n\times m}$, represents the relationship between control actions and their effects on the state space. [16, 14]

Equation 1.2 is the quadratic cost function used for the LQR controller. This function produces a singular cost value which reflects the performance of the controller based on the state trajectory, desired state value, and magnitude of controller input. An optimal control implementation of the pole-placement theory, the LQR controller abstracts the locations of the poles through the use of state and control cost matrices. The state cost matrices (Q, $Q_F \in \mathbb{R}^{n\times n}$) and control cost matrix (R $\in \mathbb{R}^{m\times m}$) assign weights to each state or control element respectively. These values can be tuned to change the relative weights of the quadratic cost function, likely producing a new optimal control sequence to minimize the total system cost. [16, 14]

$$J(\mathbf{x}, \mathbf{u}) = x_T^T Q_F x_T + \sum_{t=1}^{T-1} x_t^T Q x_t + u_t^T R u_t \tag{1.2}$$

1.2.2 LQR And Least-Squares Optimization

As seen in Equation 1.2, the LQR controller minimizes a quadratic cost function. As the quadratic cost function represents a convex optimization, the LQR controller can be solved using a weighted least-squares optimization. Prior to solving the least-squares optimization, however, the entire state and control trajectories must be embedded into a single matrix. Equations 1.3 to 1.6 specifically detail this reformatting by representing the entire state trajectory as a function of the initial state vector. [16, 14]

$$x_2 = Ax_1 + Bu_1 \tag{1.3}$$

$$x_3 = Ax_2 + Bu_2 \tag{1.4}$$

$$x_3 = A(Ax_1 + Bu_1) + Bu_2 \tag{1.5}$$

$$x_3 = A^2x_1 + ABu_1 + Bu_2 \tag{1.6}$$

Equation 1.3 is a restatement of Equation 1.1 from t=1 to t=2. Equation 1.4 additionally reflects the dynamics of the problem from t=2 to t=3. As can be seen in Equations 1.5 and 1.6, Equation 1.4 can be rewritten in terms of the initial state vector value x_1. Using the constant A and B matrices which represent the dynamics between discrete time intervals, the state vector at any discrete time step can be written in terms of the initial state vector and the history of control actions. A concise, vectorized approach of this relationship between all discrete time steps and the initial state can be seen in Equation 1.7 and simplified in Equation 1.8. [16, 14]

$$\mathbf{x} = \begin{pmatrix} 0 & 0 & 0 & 0 & \dots & 0 \\ B & 0 & 0 & 0 & \dots & 0 \\ AB & B & 0 & 0 & \dots & 0 \\ A^2B & AB & B & 0 & \dots & 0 \\ A^3B & A^2B & AB & B & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ A^{T-1}B & A^{T-2}B & A^{T-3}B & A^{T-4}B & \dots & B \end{pmatrix} \mathbf{u} + \begin{pmatrix} I \\ A \\ A^2 \\ A^3 \\ \vdots \\ A^T \end{pmatrix} x_1 \tag{1.7}$$

$$\mathbf{x} = G\mathbf{u} + Hx_1 \tag{1.8}$$

Similar to the vectorization of the state space dynamics, the cost function from Equation 1.2 is also vectorized as seen in Equations 1.9 and 1.10. Furthermore, by using Equation 1.8, the cost function can be rewritten as seen in Equation 1.11. By rewriting the full state space vector, **x**, the total cost becomes a function of only one unknown, **u**, which is the value solved for in the least-squares optimization. [16, 14]

$$J = \mathbf{x}^T \begin{pmatrix} Q & 0 & \dots & 0 \\ 0 & Q & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & Q_F \end{pmatrix} \mathbf{x} + \mathbf{u}^T \begin{pmatrix} R & 0 & \dots & 0 \\ 0 & R & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & R \end{pmatrix} \mathbf{u} \tag{1.9}$$

$$J = \mathbf{x}^T \mathcal{Q} \mathbf{x} + \mathbf{u}^T \mathcal{R} \mathbf{u} \tag{1.10}$$

$$J = (G\mathbf{u} + Hx_1)^T \mathcal{Q} (G\mathbf{u} + Hx_1) + \mathbf{u}^T \mathcal{R} \mathbf{u} \tag{1.11}$$

Having expressed the total cost as a function of the full history of control actions, Equation 1.11 can be expressed as a least-squares optimization like Equation 1.12. By completing the square in Equation 1.11, the A matrix and b vector used in Equation 1.12 are found to equal Equations 1.13 and 1.14 respectively. [16, 14]

$$\arg\min_{u} \|Au - b\|_2 \tag{1.12}$$

$$A = -(R + G^T \mathcal{Q} G) \tag{1.13}$$

$$b = G^T \mathcal{Q} H x_1 \tag{1.14}$$

Now framed as a least-squares problem, many different convex optimization solvers can be used. Some of the potential solvers include QR factorization, singular value decomposition (SVD) and pseudo-inverse. [2]

1.2.3 LQR And Markov Decision Processes

The Markov Decision Process (MDP) is a probabilistic model used for stochastic control which contains a set of possible world states, S, a set of possible actions, A, a reward function based on the state and actions, R(s,a), and a description of each action's effect on each state, T. Commonly used in the reinforcement learning and artificial intelligence (AI) communities, the MDP seeks to find the optimal policy (i.e. sequence of actions) which minimize the expected value or cost of the probabilistic model. Given the constrained states and deterministic controls of the dynamic model, the probabilistic state transition matrices can be replaced with the linear dynamics of the system. Furthermore, the MDP can be assigned the same quadratic cost function (Equation 1.15) mentioned previously for the LQR controller. [8]

In order to find the optimal control sequence of the MDP over the finite horizon, a DP approach can be used. As per Bellman's principle of optimality, "if we have found the optimal trajectory on the interval from 0, 1, ..., N by solving the optimal control problem in Equation 1.2, the resulting trajectory is also optimal on all subintervals of this interval of the form t, t+1, ..., N with $t > 0$, provided that the initial condition x_t at time t was obtained by running the system forward along the optimal trajectory from time 0." [8, 9, 1]

As seen in Equation 1.16, the LQR cost function can be written as the optimal value function used in MDPs. As such, Bellman's principle of optimality implies that the optimal policy (i.e. control values) can be found recursively by optimizing the current cost at time interval t added to the optimal value function at the next state. This relationship is explicitly detailed in Equation 1.17 where the optimal value function, or "cost-to-go" is shown in Equation 1.18. [8, 9, 1]

$$J(\mathbf{x}, \mathbf{u}) = x_T^T Q_F x_T + \sum_{t=1}^{T-1} x_t^T Q x_t + u_t^T R u_t \tag{1.15}$$

$$V^*(x_0) = \min_{u_0,\ldots,u_{N-1},x_0} J(u_0, \ldots, u_{N-1}, x_0) \tag{1.16}$$

$$u_t^* = \arg\min_{u_t} \left(x_t^T Q x_t + u_t^T R u_t + V^*(Ax_t + Bu_t) \right) \tag{1.17}$$

$$V^*(x_t) = \left(x_t^T Q x_t + u_t^{*T} R u_t^* + V^*(Ax_t + Bu_t) \right) \tag{1.18}$$

In order to solve the MDP problem, Equations 1.17 and 1.18 must be solved simultaneously, backwards in time. Ultimately, solving the MDP problem leads to the following theorem producing Equations 1.19 to 1.22 below. Equation 1.19 reflects the cost-to-go, or optimal value associated with each state-action pair along the length of the trajectory. Furthermore, Equations 1.21 and 1.22 are the Discrete Algebraic Ricatti Equations (DARE) which get solved backwards in time in order to produce the optimal control policy represented by Equation 1.20. [9]

$$V^*(x_t) = x_t^T P_t x_t \tag{1.19}$$

$$u_t^* = -K_t x_t \tag{1.20}$$

$$P_t = Q + K_t^T R K_t + (A - BK_t)^T P_{t+1} (A - BK_t), \text{ where } P_N = Q_F \tag{1.21}$$

$$K_t = (R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A \tag{1.22}$$

Figure 1.5 details a graphical representation of the Markov Decision Process. The "x" represent the states of the system over time adn the "u" represent the controls or actions taken over time. Finally, the "V" represent the next state cost to go. The graph naturally encodes the Markov Property, where each state is only affected by the previous state-action pair and not by any prior history. The graphical representation of the dynamics problem used by the MDP is further utilized by the factor graph.

1.2.4 LQR And Factor Graphs

While LQR control problems have traditionally been solved using dynamic programming (DP) approaches, recent advancements have re-framed optimal control problems as efficient inference problems [10, 15, 19, 11]. Inference problems are probabilistic in nature and seek to find the maximum a posteriori (MAP) estimate. The MAP estimate specifically maximizes the posterior density $p(X|Z)$ where X represents the states and Z represents the measurements (Equation 1.23). By exploiting Bayes' law and simplifying unnecessary terms, the MAP estimate can also be calculated using Equations 1.24 and 1.25. [4]

$$X^{MAP} = \arg\max_X p(X|Z) \tag{1.23}$$

$$X^{MAP} = \arg\max_X \frac{p(Z|X)p(X)}{p(Z)} \tag{1.24}$$

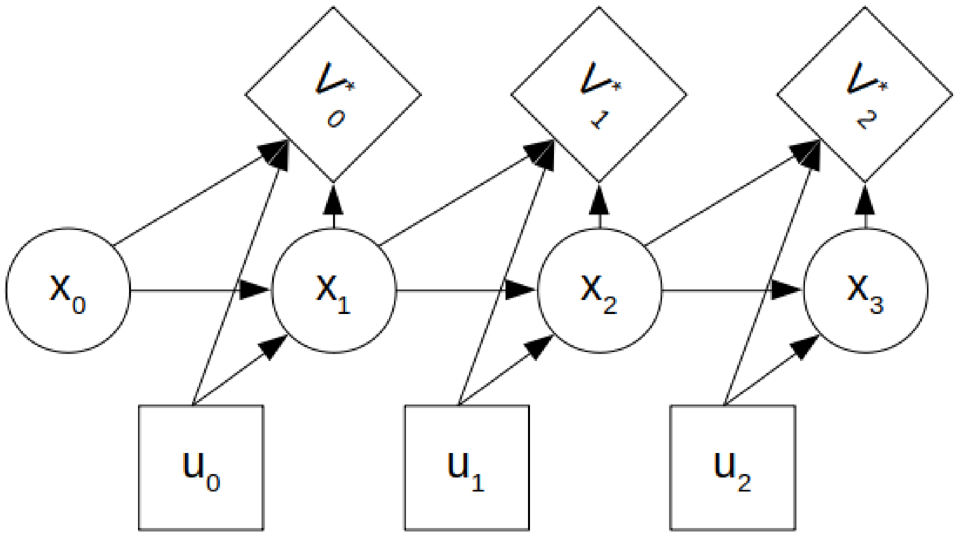

Figure 1.5: Graphical representation of a Markov Decision Process

$$X^{MAP} = \arg\max_X l(X;Z)p(X) \tag{1.25}$$

In order to solve for the MAP estimate, probabilistic graphical models (PGM) are commonly used. The PGM can represent the probabilistic nature of each variable while outlining the problem in an intuitive, graphical interpretation which commonly leverages the structural sparsity between variables. Among the many types of PGM, Bayes nets and factor graphs are two very commonly used implementations. [4]

The Bayes net is a directed graphical model that uses nodes which represent all variables of interest. Specifically the Bayes net finds the joint probability density of all of the variables by multiplying all conditional densities together (Equation 1.26). [4, 12]

$$p(\Theta) = \prod_i p(\theta_i|\pi_i) \tag{1.26}$$

While the Bayes net can be used to find the MAP estimate of an inference problem, its combination of both states, X, and measurements, Z, in its total random variable field, Θ, makes the computation sub-optimal. Consequently, Bayes nets are commonly used for generative modeling, while factor graphs are better suited for inference. [4, 12]

Factor graphs are bipartite graphs made up of factors, variables, and edges ($\mathcal{F} = (\mathcal{U}, \mathcal{V}, \mathcal{E})$). Specifically the factor graph separates unknown variables, X, from known measurements by using two separate types of nodes. Known as variable nodes and factors respectively, the factor graph explicitly models the posterior, $p(X|Z)$, rather than the joint probability, $p(X, Z)$ as does the Bayes net. Furthermore, as each factor is only connected to its corresponding node variable, the factor graph encodes a sparse factorization as seen in Equation 1.27. [4, 13]

$$\phi(X) = \prod_i \phi_i(X_i) \tag{1.27}$$

As the factor graph represents the posterior density of the inference problem, maximization of Equation 1.27 is identical to performing the MAP estimate as outlined in Equation 1.23. Consequently, the MAP estimate can be rewritten as seen in Equation 1.28. [4, 13]

$$X^{MAP} = \arg\max_X \prod_i \phi(X_i) \tag{1.28}$$

While each factor can represent any function, a Gaussian noise model is often used to represent the probability density of each variable. As seen in Equation 1.29, this noise model for the factors can then be entered into the MAP estimate equation seen in 1.28. [4, 13]

$$\phi_i(X_i) \propto \exp\{-\frac{1}{2}\left\| h_i(X_i) - z_i \right\|^2_{\Sigma_i}\} \tag{1.29}$$

After taking the negative log of Equation 1.28 and removing the constant $\frac{1}{2}$, the MAP estimate can be rewritten as seen in Equation 1.30. Instead of a maximization problem, the MAP estimate is now written as a minimization problem. More specifically, the MAP estimate is the minimization of the sum of nonlinear least-squares, where Σ_i represents the Mahalanobis norm. For the linear case, Equation 1.30 can be rewritten as seen in Equations 1.31 where A does not represent a measurement prediction function, but rather the Jacobian or system dynamics. Furthermore, the summation outside of the L2 norm can be removed and vectorized as a large matrix operation as seen in Equation 1.32. [4, 13]

$$X^{MAP} = \arg\min_X \sum_i \left\| h_i(X_i) - z_i \right\|^2_{\Sigma_i} \tag{1.30}$$

$$X^{MAP} = \arg\min_X \sum_i \left\| A_i(X_i) - z_i \right\|^2_{\Sigma_i} \tag{1.31}$$

$$X^{MAP} = \arg\min_X \left\| AX - \mathbf{z} \right\|^2_{\Sigma} \tag{1.32}$$

While not addressed in this report, nonlinear factor graphs must then solve the least-squares minimization incrementally by using a nonlinear solver such as steepest descent, Gauss-Newton, Levenberg-Marquardt, or Powell's dogleg minimization. [4]

As each factor of the factor graph corresponds to a block row of the Jacobian matrix, A, the factor graph naturally encodes the least-squares objective (or alternatively, unnormalized posterior). By performing a process known as variable elimination, the factor graph can be converted into a Bayes net based on the unknown variables X. Given the parent-node relationship of the Bayes net, the MAP estimate can then be easily found by performing back-substitution on the Bayes net. [4, 13]

Figure 1.6 shows how a factor graph can be used for LQR control. In this example, there is a trajectory of three states, x, and two control inputs,

u. Both the states, x, and controls, u, are unknown and consequently represented using variable nodes. Connecting the variable nodes are two different sets of factors. The first connect the states together through time. These represent the dynamics of the problem, in the form $x_{t+1} = Ax_t + Bu_t$, and are alternatively known as hard constraints. Additionally, there are unary factors connected to only the state variables and only the control variables. These are cost minimization factors and represent the control and state cost errors used with LQR control. In this example, the cost minimization factors impose priors on the variable nodes of the form x^TQx and u^TRu for the state and control nodes respectively. [6]

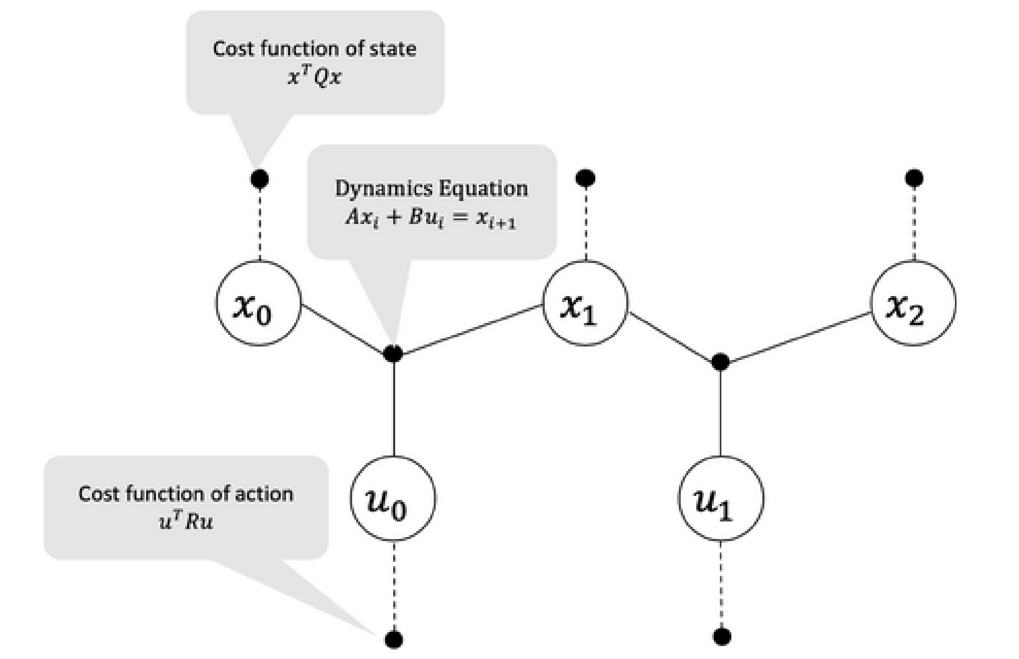

Figure 1.6: Using a factor graph to represent a constrained least-squares problem [6]

A more complete example of the factor graph implementation of the LQR controller can be seen on the left in Figure 1.7. In this example, the variable elimination algorithm starts at variable node x_2. As per the variable elimination algorithm, the first step in eliminating the node is to select all factors immediately connected to the node of interest and the other nodes additionally connected to them. In this case, the red edges and factors define the separator for variable node x_2. [4, 6]

In order to successfully convert x_2 factors into a Bayes net format, the factors are combined. In this example, the two factors combined represent constraints between three variable nodes: x_1, x_2, u_1. The goal of variable elimination is to rewrite the combined product factor as a node-parent Bayes net relationship and a new factor which is a function of all nodes, except the current elimination node (i.e. x_2). As can be seen on the right side of Figure 1.7, the two red factors on x_2 are converted into a new blue factor between variable nodes x_1 and u_1 and a Bayes net from variable nodes x_1 and u_1 to x_2. The constraints experienced by variable node x_2 have now been encoded into a Bayes net and the other nodes involved in said constraints have rewritten the factor to reflect the change in topology of the graph. [4, 6]

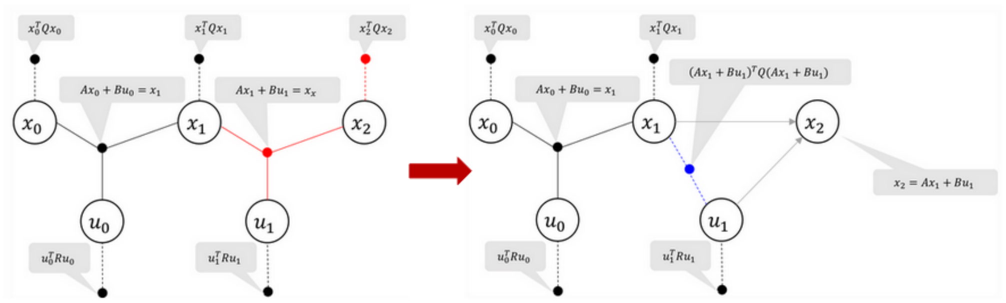

Figure 1.7: Eliminating state of factor graph [6]

Ultimately, this process of variable elimination models the chain rule. As the factor graph naturally encodes sparsity with its relationships, however, the variable elimination algorithm is computationally much faster. As seen in Figure 1.8, the process will continue until the whole graph has been converted from a factor graph to a Bayes net written with respect to the unknown variables, x and u. As previously mentioned, the Bayes net format is then optimal for back-substitution in order to compute the MAP estimate. [4, 6]

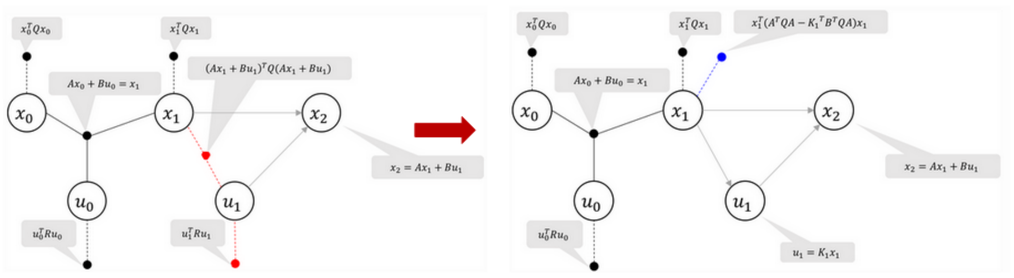

Figure 1.8: Eliminating control of factor graph [6]

Chapter 2

Research Question

This research explores three fundamental questions. First, we determine if an LQR controller designed using a factor graph provides substantial improvement over existing LQR implementations for real-time flow control of a wireless mesh network. Second, we examine if physical RF links between communication nodes can be used in the solving of the factor graph in order to make the controller more robust and decentralize the LQR controller design. Last, we wish to investigate the scenarios where the WMNLQR improves network flow control in comparison to traditional heuristic methods (i.e. fixed rate, etc) for scenarios comparable to those faced in the DARPA SubT Challenge.

Chapter 3

Related Work

[4] showed the applicability of using factor graphs to perform inference in large problems. The work begins with the discussion of Bayes nets and their functionality for inference problems. As the Bayes net treats both the unknown states, X, and measurements, Z, as random variables, the Bayes net becomes suboptimal in solving large inference problems. Instead, the Bayes net is great for generative modeling and ancestral sampling.

[4] then show that the weakness of the Bayes net with respect to inference can be solved by factor graphs. Specifically it is shown that the factor graph is a natural representation of the posterior density of a problem, $p(X|Z)$, instead of the joint probability, $p(X, Z)$, found with Bayes nets. This is further clarified as it is illustrated that the bipartite nature of the factor graph helps isolate factors (i.e. measurements) from unknown variables (i.e. states).

Next, [4] outline a common factor graph representation - the Gaussian factor graph. It is shown that the Gaussian factor graph, when treated independently, is equivalent to a least squares optimization. When combined with an iterative nonlinear optimizer, nonlinear factor graphs (or nonlinear least squares) problems can additionally be solved.

In order to perform the inference, the factor graph can be converted into a Bayes net that only models the unknown variables, X. This process is described as variable elimination and specifically detailed in summaries of the algorithms. The algorithms illustrate that the factor graph is incrementally converted into a Bayes net using an efficient version of the chain rule. Having then converted the factor graph into a Bayes net, a process known as back substitution takes place in order to output the MAP estimate of the original inference problem.

In addition to the detailing of the factor graph solver, [4] illustrate the importance of variable ordering on the solver's efficiency. Furthermore, the factor graph inference problem is regularly extended to SLAM or state estimation problems.

[20] presented an algorithm known as Equality-Constrained LQR (EC-LQR). As the LQR problem is identical to a least squares optimization, the algorithm can use factor graphs to calculate the LQR control. The work is inspired by the increase in use of probabilistic graphical models (PGM) and factor graphs in the state estimation and SLAM communities.

Additionally, [20] use the factor graph in order to impose additional local equality constraints and cross time-step equality constraints. Such constraints could potentially resemble synchronous motion exhibited by a robot (i.e. the contact of a foot with the ground for a walking robot).

In order to better illustrate the variable elimination process for the LQR controller using factor graphs, both the normal factor graph and EC-LQR version show examples of both the states and controls being solved with variable elimination.

When testing the performance of the EC-LQR, it is shown to achieve comparable cost to other quadratic programming benchmark algorithms for a simple problem with local constraints. The addition of the auxiliary constraints, however, only imposes a linear growth in runtime while the dynamic programming approaches which experience cubic growth.

Finally, [20] finish with a cross time-step constraint example between EC-LQR and two baseline implementations. A perturbed initial state is presented to a walking robot control problem. The EC-LQR is capable of producing an optimal control sequence while the other baseline implementations, including MATLAB's quadratic programming algorithm, are unable.

[17] presented an algorithm known as Sparse Graphical Optimizer (SGOPT) which utilized factor graphs for optimal control. The goal was a detailed analysis of the factor graph LQR implementation in comparison to two dynamic programming LQR approaches. The specific dynamics problem on which the controllers were based was the cart-pole problem where carts were inter-connected with springs and dampers.

The results illustrate the linear runtime complexity of the factor graph solver with respect to trajectory length and state space size. Meanwhile the dynamic programming approaches (including the highly optimized Control Toolbox implementation) were still susceptible to cubic runtime growth with respect to state space size.

[17] concludes with introductory results to a nonlinear dynamics implementation, where the factor graph is coupled with a Levenberg-Marquardt optimizer.

[5] look at using factor graphs to perform both state estimation and deterministic control for a UAV obstacle avoidance task where there exist nonlinear dynamics. This LQG implementation helps express the duality of estimation and control as optimal problems.

[5] leverage factor graphs to be a unifying framework for both the estimation and control problems. They also show a factor graph implementation of the Sequential Quadratic Programming (SQP) algorithm which was needed to optimize the factor graph given the presence of nonlinear constrained factors which Levenberg-Marquardt and Gauss-Newton are unable to optimize. [5] conclude their work with the deployment as an MPC controller for the UAV obstacle avoidance task.

Chapter 4

Methods

4.1 WMNLQR Versus Other LQR Solvers

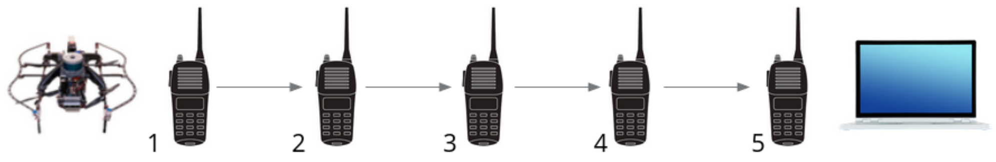

Figure 4.1: Example communication node topology with 5 nodes in a linear chain

In order to test the performance of the different LQR implementations a linear chain of nodes, like Figure 4.1 was used. While WMN typically have a 2-dimensional topology, a 1-dimensional topology simplified illustration and preliminary analysis. As such, the nodes followed the dynamics model shown in Equations 4.1 to 4.5.

For this experiment, the state space included the current size of the communication node's outbound queue and the rate at which it was transmitting traffic. The state dynamics for nodes outputing data is modeled by Equation 4.2. Each node's queue is appropriately updated by the rate at which it outputs its traffic. Furthermore, each node's queue is also updated by the rate at which traffic flows into the node. This is represented by Equation 4.3. Finally, the control elements are shown in Equations 4.4 and 4.5. The control element is the change in the output rate of a node as it transmits traffic.

$$x = \begin{bmatrix} \text{Queue Size} \\ \text{Rate} \end{bmatrix} \tag{4.1}$$

$$A_{out} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \tag{4.2}$$

$$A_{in} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{4.3}$$

$$B = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{4.4}$$

$$u = \begin{bmatrix} \Delta Rate \end{bmatrix} \tag{4.5}$$

As not all nodes both send and receive traffic, Equations 4.6 and 4.7 illustrate the two different node scenarios for this experiment. For simplicity it is assumed that node 5 both receives data from node 4 and sends data to the laptop computer. Equation 4.6 represents the dynamics experienced at node 1. Equation 4.7 represents the dynamics experienced at node 3.

$$x_{t+1}^1 = A_{out} x_t^1 + B u_t^1 \tag{4.6}$$

$$x_{t+1}^3 = A_{out} x_t^3 + A_{in} x_t^2 + B u_t^3 \tag{4.7}$$

Node 1, as seen in Equation 4.6, only outputs traffic and uses the output dynamics A matrix. All other nodes, however, both send and receive data. Consequently, their state vector is a function of their internal state dynamics but also the flow rate of adjacent nodes. Equation 4.7 better illustrates this concept by showing that the state at node 3 is a function of its own output rate, but also the output rate from node 2.

In order to test the performance of the LQR implementations as the total state and control spaces changed in size, the number of nodes in the linear chain increased from 5 to 50 in intervals of 5.

Five different LQR implementations were tested in C++ on this linear chain of nodes. The first two were least-squares approaches. Using the C++ Eigen library, the least squares problem was solved using both QR factorization and singular value decomposition (SVD). The next two were dynamic programming approaches. The first was a standard Discrete Algebraic Ricatti Equation (DARE) with partial-QR factorization designed using Equations 1.19 to 1.22. The second was the Control Toolbox implementation of the LQR controller from ETH Zurich [7]. This implementation has a highly efficient C++ backend which makes it a very effective dynamic programming approach. Last, the factor graph implementation was designed using the Georgia Tech Smoothing And Mapping (GTSAM) library [6].

The Gaussian factor graph was used to model the linear dynamics of the network control problem. A small representation of the factor graph for the linear chain of wireless nodes can be seen in Figure 4.2. The numbered light grey nodes each represent a communication node laid out spatially in the x-axis. The y-axis, or time dimension, represents the trajectory of each node across discrete time intervals. For illustration, this figure also includes a dark grey node labeled 'u'. This node represents a control node used to influence the dynamics experienced at communication node 3. The simulation results displayed later include more than one controllable node.

As seen in the attached legend, the red squares are factors which represent dynamics constraints between nodes. For the case of the dynamics model selected for the WMN, these hard constraints contain both spatial and temporal dependencies between communication nodes. Furthermore, the orange squares are prior factors which represent the initial conditions of each communication node at the beginning of the trajectory. These factors are additionally treated as hard constraints. Next, the green and yellow squares represent soft constraints, or optimization objectives. The green factors represent the cost of each control action while the yellow factors represent the cost of each variable state from the desired final value. The objective in solving the factor graph is to minimize the soft constraints provided the dynamics imposed by the hard constraints.

In order to solve the factor graph, a COLAMD variable elimination ordering was used. As the variable elimination algorithm removes factors, the factor graph converts to a Bayes net. Once the entire factor graph has been converted to a Bayes net, the initial conditions provided by the prior state factors can then be used to perform a backward substitution and output the optimal control policy, u^*, and corresponding state trajectories, $\mathbf{x}$, over the finite horizon.

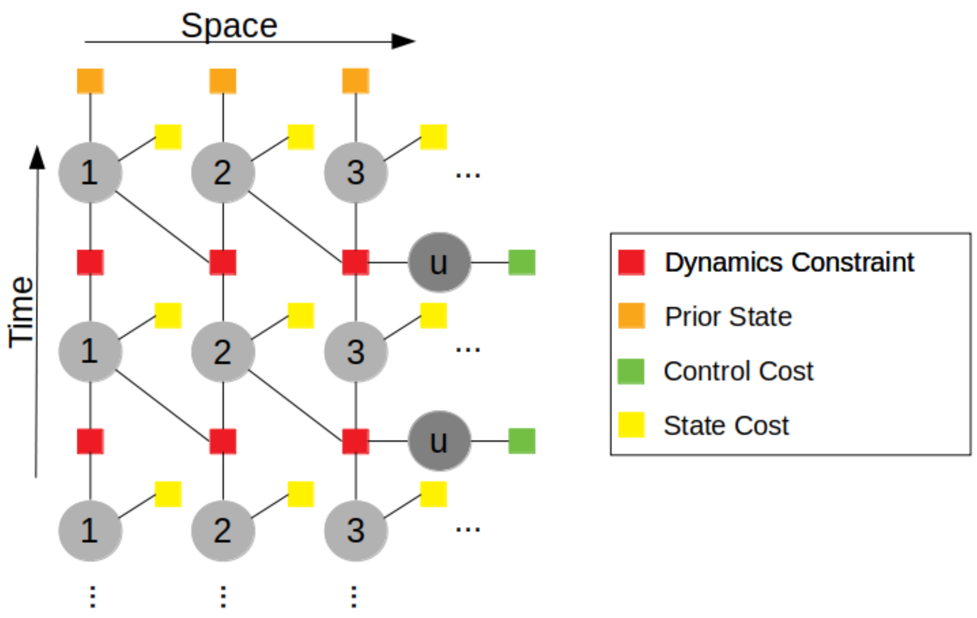

Figure 4.2: Example factor graph topology for a linear chain of 3 wireless nodes across 3 time steps

In order to monitor the performance of each LQR implementation, the solve time, total cost, state trajectories and control trajectories were captured for each run. A more detailed comparison between each LQR implementation will be outlined in the Results section.

4.2 Decentralizing the WMNLQR

The WMNLQR models a unique dynamical problem where the factor graph nodes represent wireless communication nodes. As such, there are physical message passing capabilities between nodes (i.e. RF message passing) which can supplement the traditional message passing performed when solving a factor graph

Now traditionally, decentralizing or decoupling controllers will yield sub-optimal global solutions. By leveraging the RF message passing capabilities, however, the equivalent global factor graph can still be solved by subdividing it into a series of subproblems.

Figure 4.3 begins to illustrate the decentralization of the controller. The leftmost section of the figure shows the same factor graph topology as seen in Figure 4.2. Specifically, three communication nodes are placed in a linear chain where only node 3 is controllable. Additionally, a finite horizon of three time steps is used for this simplified scenario.

Focusing first on converting node 3 and its control element, "u", into a Bayes net, the global factor graph can be rewritten as seen in the middle figure of 4.3. As node 3 is only dependent on node 2 in this scenario, there is no need to incorporate node 1 states or constraints. To rephrase, node 3 is only dependent on the dynamics and conditions of its connected nodes (e.g. node 2).

The middle figure can be further simplified. Because node 1 has been eliminated from the factor graph, the dynamics at node 2 are now corrupted and ill-formed. As node 2 will be solved later, however, its dynamics do not have to be perfectly accurate. Instead, the only dynamics involving node 2 which must be accurate are its dynamical relationship with node 3. As a result, the dynamics (red) factors between all time steps at node 2, as well as, the prior (orange) and state cost (yellow) factors can be removed from the factor graph representation. They can then be replaced with a single arbitrary factor of equal size to the state space. This is illustrated by the black squares in the rightmost section of Figure 4.3.

As the state values at node 2 will be provided in the back substitution process, the arbitrary constraint can be treated as a prior factor which has yet to be given any value. Ultimately, this arbitrary factor encodes all constraints that would nominally be imposed on node 2. By simplifying the dynamics and unknowns to a single arbitrary constraint, however, node 3 no longer requires any knowledge of node 2 or its own respective dynamics. Consequently, the decentralized solver can create and solve a subproblem only in terms of the nodes and dynamics specifically connected to node 3's finite horizon.

Having restructured the global problem into a local problem at communication node 3, the new factor graph can now be converted into a partial Bayes net. This can be seen in the leftmost section of Figure 4.4. In this example, the factor graph is converted into a Bayes net only at communication node 3 and its corresponding control elements. Furthermore it is solved backwards in time from t=N to t=1, where N is the trajectory length. Technically the variable elimination could be performed using CO-

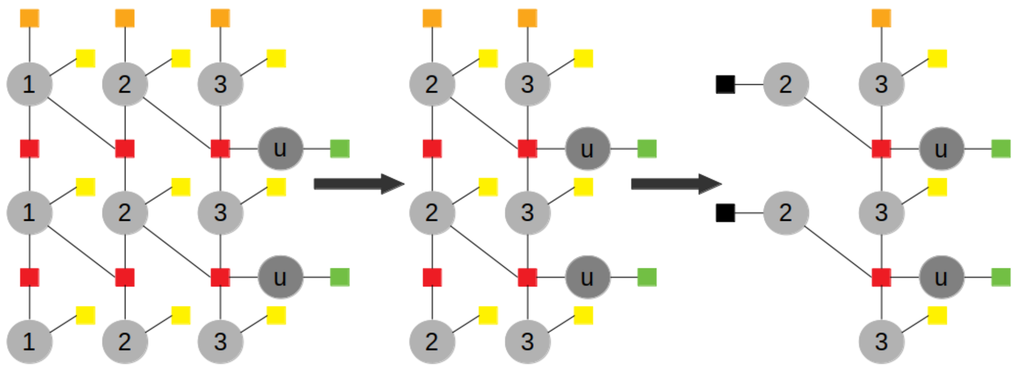

Figure 4.3: The process of reformatting a global factor graph into a sequence of sub-topologies

LAMD or any arbitrary ordering. However, the factor graph must only convert node 3 and its control elements to a Bayes net while leaving all other nodes (i.e. node 2) in factor graph representation.

Once the full factor graph at node 3 has been converted into a Bayes net, a new factor (as seen by the purple square in the leftmost section of Figure 4.4) is attached to the variables nodes for communication node 2. This factor encodes all of the hard and soft constraints present in the subproblem at communication node 3 which have now been message passed to the variable nodes at communication node 2. A cleaner representation of the partial factor graph and Bayes net can be seen in the middle graphic of Figure 4.4.

The decentralized controller running at communication node 3 has now converted its modified factor graph topology into a Bayes net at node 3 and a new factor. The controller now saves the Bayes net and passes the new factor node as a physical RF message to communication node 2. Communication node 2 can now create its subproblem in a way similar to node 3 as seen in Figure 4.3. As node 2 is only connected to node 1 (node 3 was already converted), it can use arbitrary constraints to tie down node 1 and it can produce the factor graph seen in the rightmost graphic of 4.4. Having leveraged its RF capabilities, however, node 2 also attaches the purple factor passed from the decentralized controller running at node 3. Consequently, node 2's factor graph now encodes all of the information present within communication node 3's factor graph, although neither communication node was fully aware of the dynamics of the other node. This forward process then iterates with solving subproblems and RF message passing constraints until all communication nodes have had their local dynamics converted to Bayes nets.

Once the forward substitution process has been completed for all communication nodes, each node will have saved a Bayes net describing the dynamics of its local subproblem. As illustrated in Figure 4.5, the Bayes net at communication node 1 (leftmost graphic) can now be solved using the prior constraints its factor graph used. Now, the entire state trajectory at communication node 1 has been solved. By revisiting communication

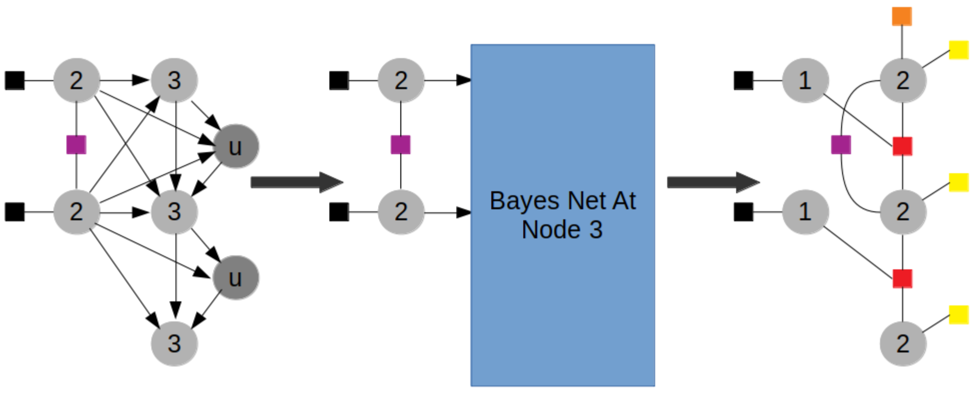

Figure 4.4: The forward substitution process for the DWMNLQR controller

nodes in the reverse order of the forward pass, communication node 1 can now RF message pass its state trajectory to communication node 2. This state trajectory can be treated as input to the Bayes net stored by the decentralized controller at communication node 2 (rightmost graphic in Figure 4.5). Communication node 2 may now perform a local backward substitution and solve the entire finite horizon trajectory at communication node 2. This process of forward substitution to solve the Bayes net and RF message pass the state trajectory to the next node then iterates until all communication nodes have had their local Bayes nets solved.

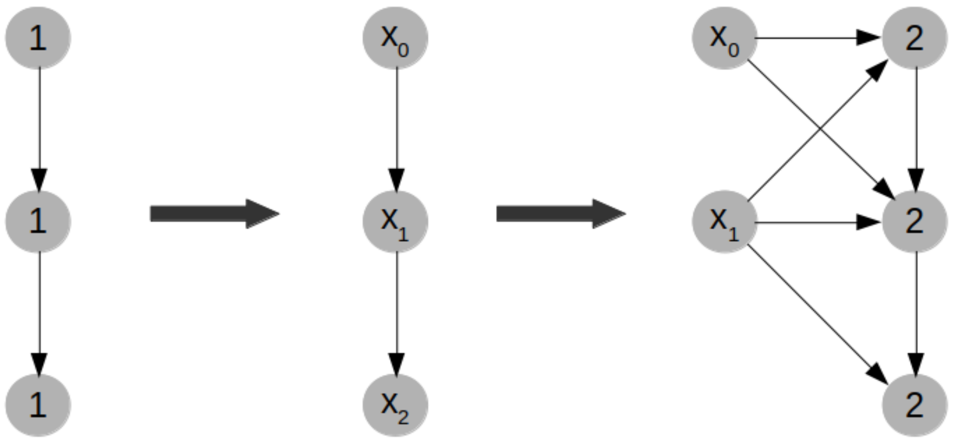

Figure 4.5: The backward substitution process for the DWMNLQR controller

Chapter 5

Results

5.1 WMNLQR Versus Other LQR Solvers

The first experiment outlined in the Methods section was the comparison of the WMNLQR controller to other LQR solvers. As previously mentioned, a linear chain of wireless communication nodes was used to model the problem dynamics. In an attempt to compare the different LQR implementations as the state and control spaces changed in size, ten different experiments were run where the linear chain grew in size from 5 communication nodes to 50 in intervals of 5. Furthermore, the same 5 communication nodes were controllable in each of the ten problems and the trajectory length was 30 time steps.

Prior to comparing the computational performances of each implementation, first, the accuracy of the WMNLQR as an LQR implementation was verified. As seen in Figure 5.1, all five LQR implementations achieved identical cost for each of the ten problems.

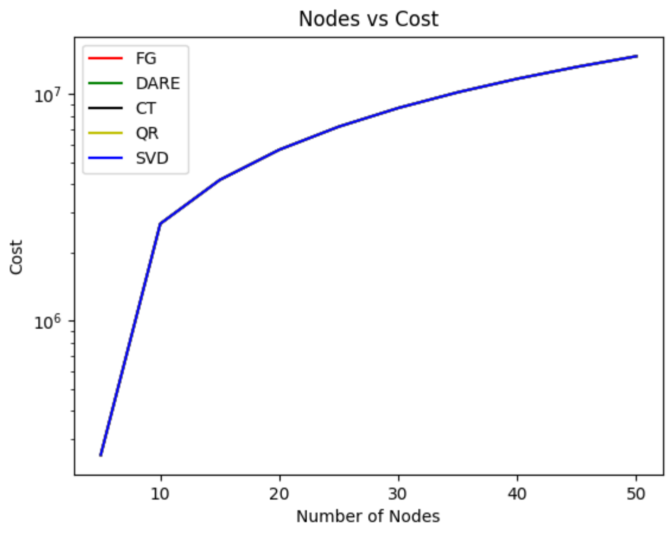

Figure 5.1: The optimized cost values of 5 different LQR solvers as the number of communication nodes in the problem increases from 5 to 50

Having validated the legitimacy of solving an LQR problem using inference and factor graphs, the next step was to illustrate the performance improvements when using WMNLQR. Figure 5.2 shows the runtime to compute the optimal controls for the entire factor graph using each LQR implementation. Immediately apparent is the dramatic increase in time it takes to solve the two least-squares implementations of LQR (QR, SVD) over the three dynamic programming methods (FG, DARE, CT). From Figure 5.2 the least-squares approach show a cubic rate of growth as the state space increases in size, while the dynamic programming approaches appear linear.

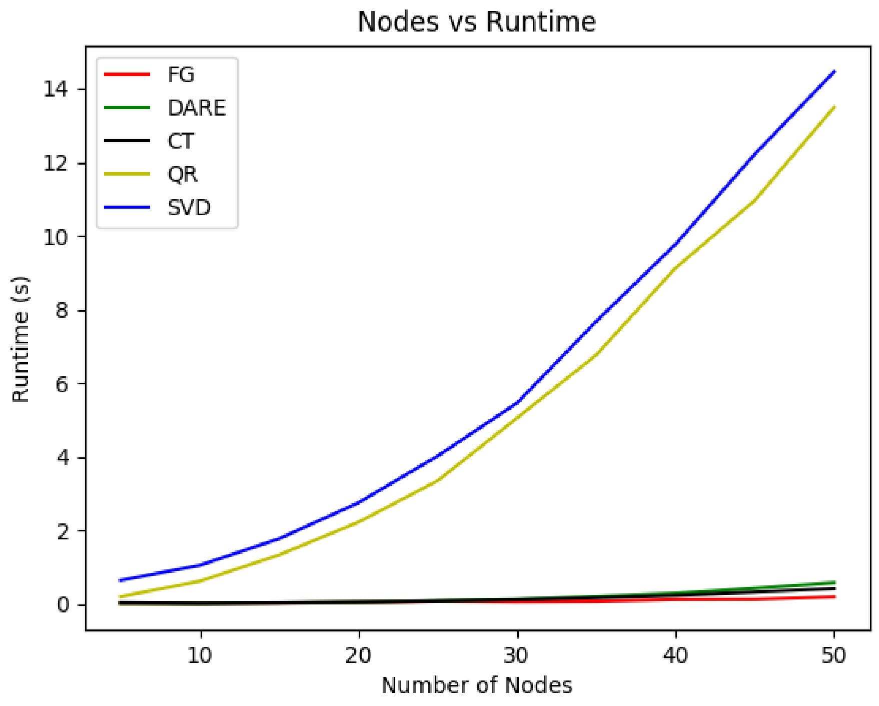

Figure 5.2: The runtime of 5 different LQR solvers as the number of communication nodes in the problem increases from 5 to 50

While the three dynamic programming methods look fairly comparable in runtime performance, there is a noticeable difference in their capabilities. Figure 5.3 better illustrates the variation in performance by zooming into the plots of the three dynamic programming methods.

As expected, both the C++ DARE implementation and the Control Toolbox [7] implementation experienced cubic growth in runtime as the state space increased in size. Meanwhile, the factor graph implementation, WMNLQR, experienced linear growth in runtime as the state space increased in size. Although all three dynamic programming approaches produced solutions substantially faster than the least-squares implementations, the discrepancy between the three dynamic programming algorithms becomes very clear as the number of communication nodes grows. Consequently, the presence of more nodes or larger state space representations still have the potential to make dynamic programming implementations intractable for realtime solutions.

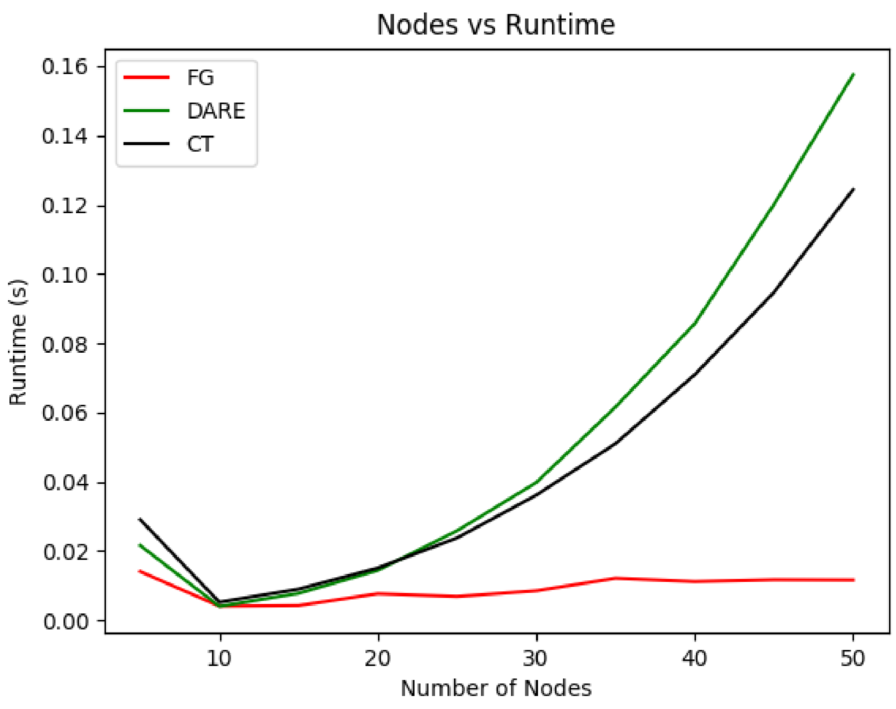

Figure 5.3: The runtime of 3 different LQR solvers as the number of communication nodes in the problem increases from 5 to 50

5.2 Decentralizing the WMNLQR

5.2.1 Validating the DWMNLQR

Having proved that the WMNLQR is capable of achieving identical cost, state trajectories, and control trajectories as all other LQR implementations, the next step was decentralizing the controller. As previously mentioned in the Methods section, the WMNLQR was converted into DWMNLQR by leveraging the RF capabilities of the communication nodes to perform physical message passing.

The first task in creating the DWMNLQR was validating whether it produced solutions identical to the WMNLQR. Specifically this was determined by comparing the state trajectories and control trajectories between both implementations on the factor graph seen in Figure 5.4 for 20 timesteps.

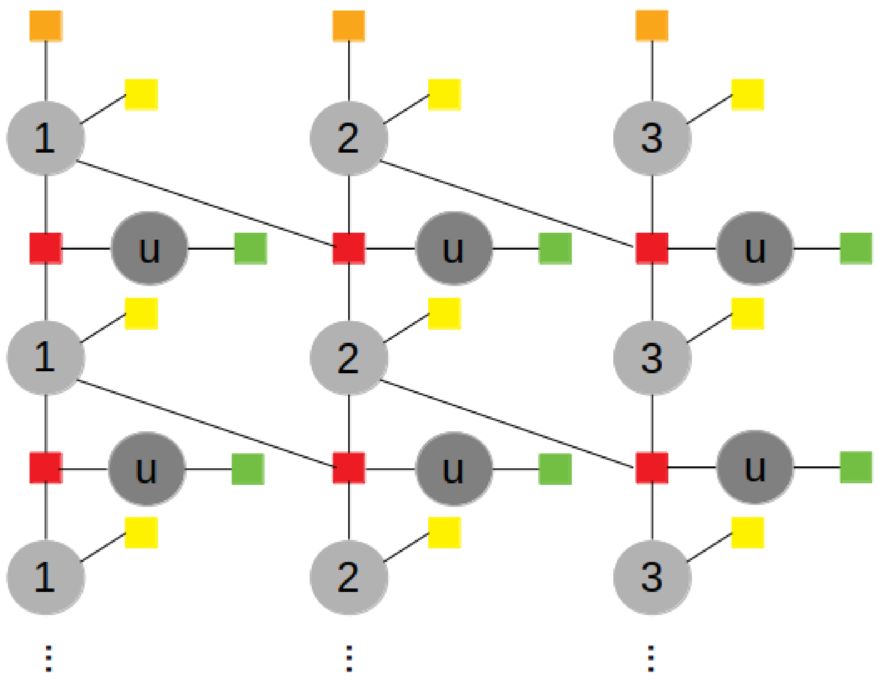

Figure 5.4: Factor graph used for comparison of WMNLQR and DWMN-LQR implementations

Figure 5.5 shows that both the centralized (WMNLQR) and decentralized (DWMNLQR) implementations achieved identical state trajectories for all three communication nodes for both the queue and the flow rate. This is further supported by the identical control trajectories for all three nodes generated by both implementations as seen in Figure 5.6.

The equivalence of the decentralized implementation should not be surprising as it is really just a partitioned global factor graph with a unique variable ordering. The equivalence does validate, however, that the global factor graph can still be optimized when each communication node is only aware of its immediate local topology.

5.2.2 Benefits of DWMNLQR

The real advantage in distributing the LQR controller design can be seen in Figures 5.9 and 5.10. When solving a factor graph for a wireless mesh network, only part of the time required to control the large network of wireless mesh nodes is used to solve the factor graph. There will also be a substantial time delay between nodes due to hardware inefficiency and latency between the radio repeaters.

Given the very fast computation of the WMNLQR implementation, in many scenarios, the ping time between communication nodes might be a bigger barrier to real-time control than the solving of the global problem itself. It is for this reason, that unnecessary point-to-point messages must be minimized between the communication nodes.

The traditional centralized approach, WMNLQR, is shown in Figure 5.7. For this example, the centralized computation node is the node sur-

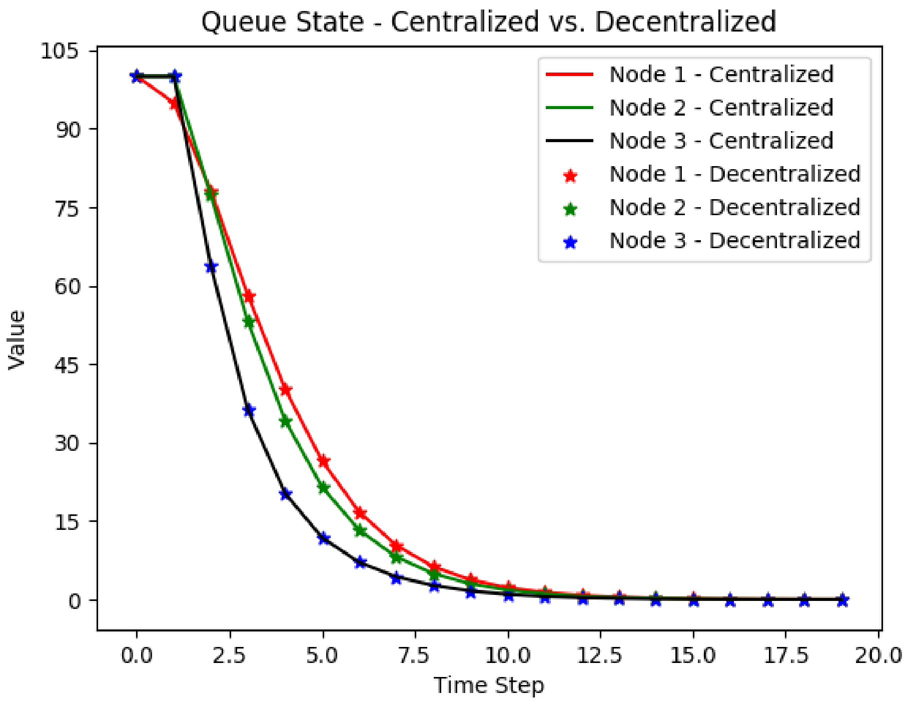

Figure 5.5: State trajectories for WMNLQR and DWMNLQR implementations on three communication node factor graph

rounded by the red box. Before generating each trajectory update, the centralized node will receive updates via RF messages from the surrounding nodes such as changes in topology, initial states, goal states, or cost function parameter tuning. Once the centralized node is fully aware of the global problem, the factor graph will be created and solved. After computing the solution, the node will now have to distribute this information back to the surrounding communication nodes. This will require more RF messages so that all nodes are ready to perform the necessary control over the length of the computed trajectory.

The decentralized approach, DWMNLQR however, will behave as seen in Figure 5.8. For this example, the first subproblem will be converted to a factor graph at node N and then the RF message passing will iterate down the chain of nodes ultimately to node 1. The process will then reverse, where the Bayes nets will be solved and the RF message passing will move the array of solutions to the next node.

As the linear chain of nodes increases from 5 to 50, the number of messages being sent between pairs of communication nodes for DWMNLQR increases linearly. For example, node 50 will send one message of data to node 49. Node 49 will then perform its computation and send one message of data to node 48. This process will continue, where the previous message is used in solving the new subproblem and ultimately abstracted into the next singular message pass.

For WMNLQR, however, this number of messages becomes exponential as the network continues to grow in size. For example, if node 15 is

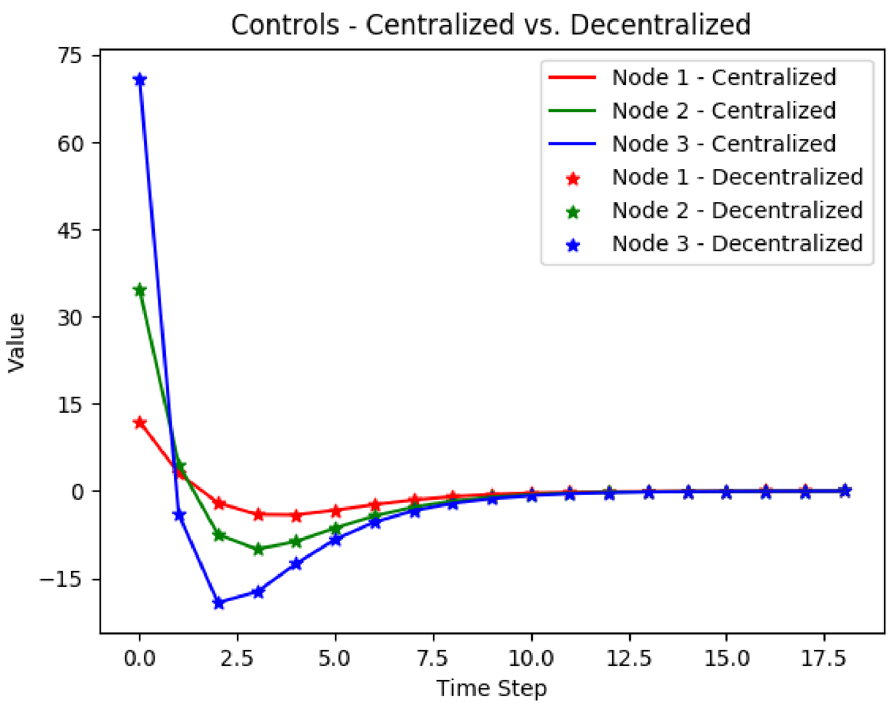

Figure 5.6: Control trajectories for WMNLQR and DWMNLQR implementations on three communication node factor graph

the centralized computation node, it will receive one message from node 14 and one message from node 16. It will also receive one message from node 13 and node 17, however, these messages must travel from two hops away. That means these messages can be treated as two message-links each. The total number of message-links continue to grow faster as the surrounding nodes grow farther from the centralized computation node. The problem is then further exacerbated on the return trip, where the centralized computation node must relay the solved control trajectories to each of the nodes. Overall, a simulation of the two relationships is demonstrated in Figure 5.9

A reiteration of this data can be seen in Figure 5.10. This graphic the runtime needed to solve the global problem after taking into account communication delays between nodes. For both simulations, the communication delay was treated as Gaussian noise with a mean of 5ms and standard deviation of 2.5ms.

Unsurprisingly, Figure 5.10 shows that WMNLQR increases exponentially in runtime, due to its inefficiency in evading the induced cost of ping time delays between communication nodes. Meanwhile, the DWMNLQR, grows at a mostly linear rate with some mild fluctuations.

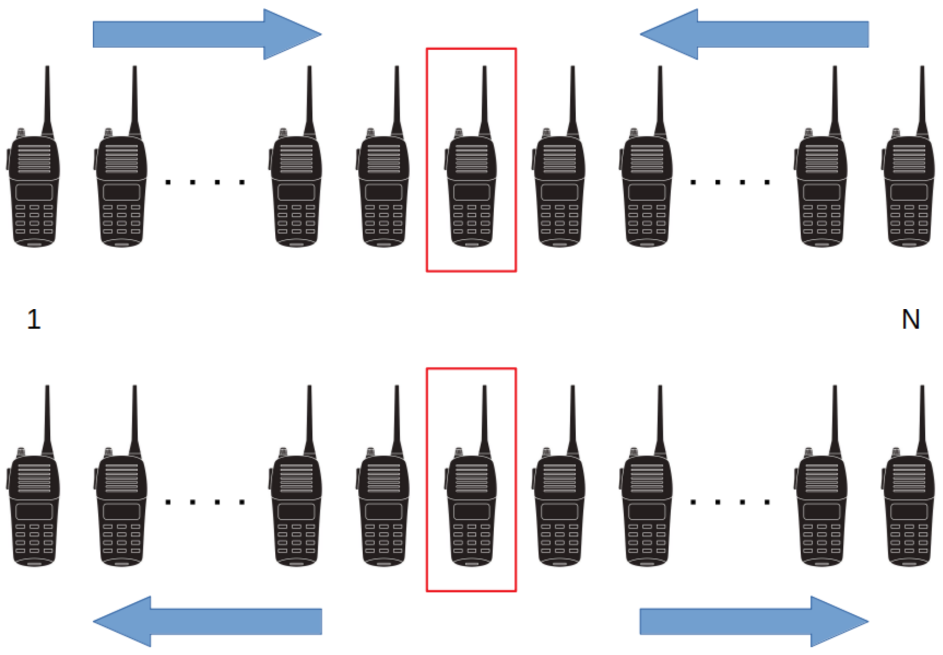

Figure 5.7: Message passing strategy to solve WMNLQR

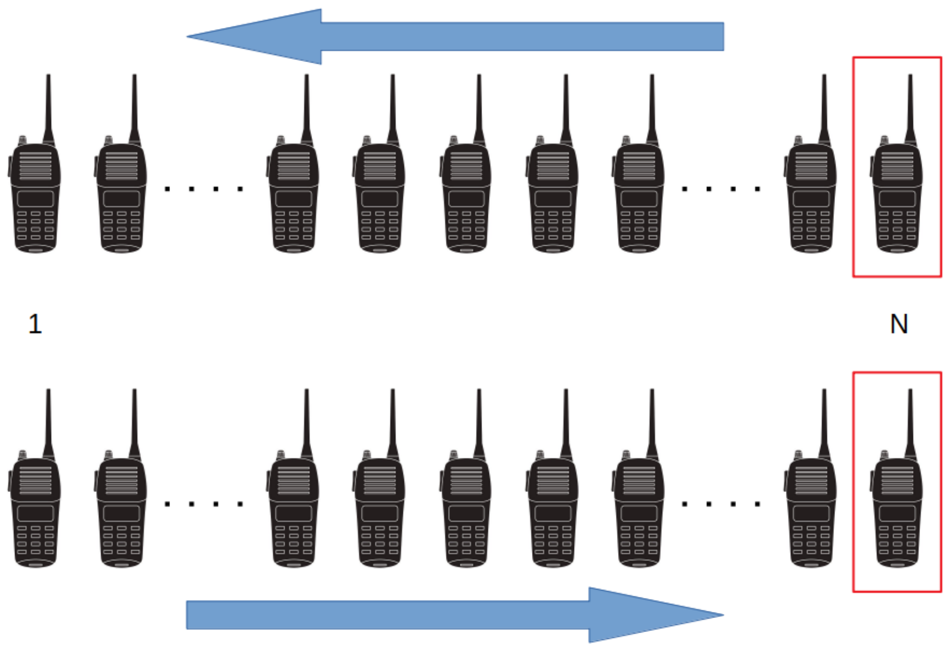

Figure 5.8: Message passing strategy to solve DWMNLQR

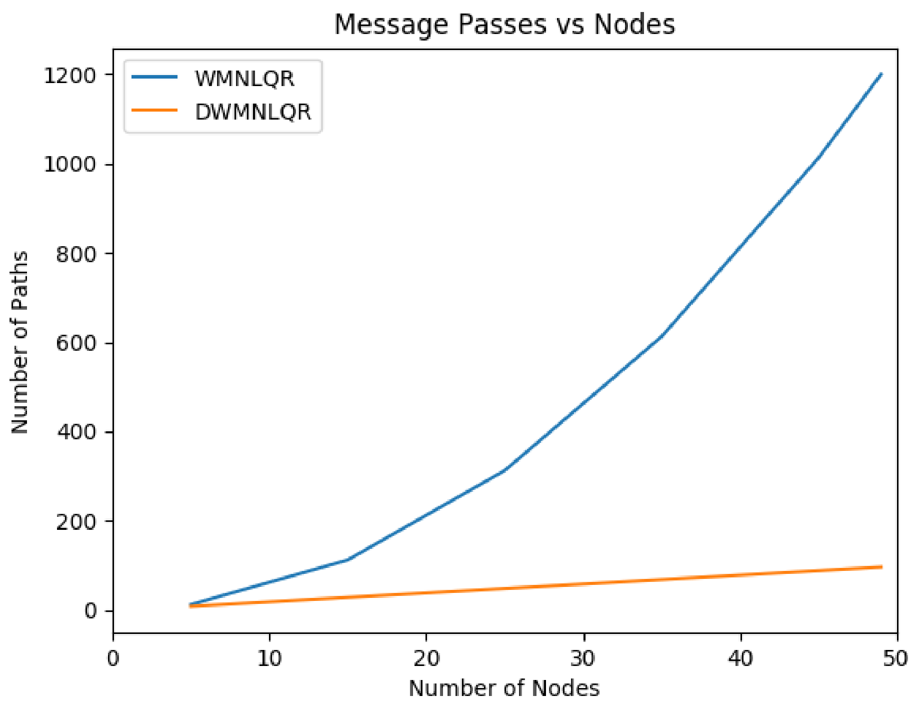

Figure 5.9: Number of messages passed in solving factor graph for WMN-LQR and DWMNLQR implementations as factor graph increases in size

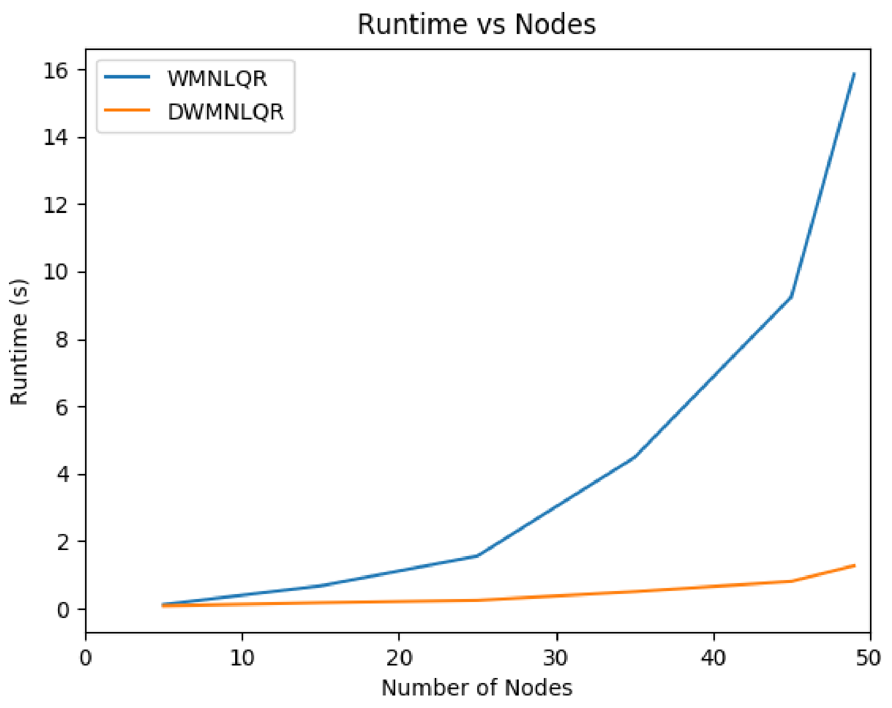

Figure 5.10: Runtime to solve factor graph for WMNLQR and DWMNLQR implementations as factor graph increases in size

Chapter 6

Conclusions

Overall, our work validated the applicability of a factor graph LQR implementation for realtime flow control of a wireless mesh network. First, we were able to create a dynamic model which represented the dynamics of each communication node in the WMN. Furthermore, we then verified that this dynamical model was controllable by LQR and could be appropriately modeled using inference and factor graphs.

Second, we outlined the clear performance benefits of using a factor graph LQR implementation over least-squares and dynamic programming approaches. Having used both math and simulation results to illustrate our point, we confirmed that the WMNLQR is capable of achieving compute time results of complexity $\mathcal{O}(T(M+N))$, where T is the trajectory length, M is the total control space dimension and N is the total state space dimension. Meanwhile, we showed that the dynamic programming approaches have compute time complexity of $\mathcal{O}(T(M+N)^3)$ and the least squares approaches have compute time complexity of $\mathcal{O}(T^3(M+N)^3)$. We also showed that the factor graph LQR implementation achieved the same cost and state and control sequences as did all other implementations, despite the dramatic runtime improvements.

Third, we exploited the confluence between wireless mesh networks and factor graphs. By leveraging the RF capabilities of communication nodes within the WMN, we created a decentralized implementation of WMNLQR. DWMNLQR is capable of achieving the same global minimum control sequence, as does WMNLQR, but without the need for a central operating node fully aware of the dynamics of the entire system. By only relying on the local topologies of each node, DWMNLQR substantially improves system robustness and runtime for the scenarios presented by real world hardware, where ping time and latency exist. DWMNLQR achieves this improvement in performance by minimizing unnecessary point-to-point messages that hinder the computation speed and reliability of centralized approaches in non-simulation environments.

Chapter 7

Future Work

In order to get a better understanding of the performance of WMN-LQR and DWMNLQR as network flow controllers, future work should prioritize both simulation and hardware testing against heuristic network flow control approaches. Specifically, the testing should be two-fold. First, the testing should attempt to find the communication scenarios (topology and traffic) where an LQR controller of the entire network performs better than a fixed rate or token bucket heuristic. Second, the testing should validate that both the WMNLQR and DWMNLQR are capable of realtime operation given the current dynamics models.

Future work will also likely modify the dynamics of the problem. Among the many considerations, the state space might consider ping time between nodes, line-of-sight, or even the location and velocity of robots (i.e. data producers). Given the high dimensionality of multiple robots exploring around an ad-hoc wireless mesh network, there are many states that affect the communications network which could be captured in the dynamics model. Furthermore, a more precise nonlinear model of the queue dynamics of each node could be used to better capture the hardware switching speed of each transceiver node.

Remodeling the dynamics also introduces the possibility of creating an LQG controller [5]. Alternatively known as a partially observable Markov decision process (POMDP), the LQG would include the LQR controller for the new dynamics plus a Kalman filter which could estimate state values based on noisy dynamics and measurements. For example, the Kalman filter could produce an optimal estimate for the ping time between nodes which is often coupled with white noise.

Another major consideration for future work is the addition of inequality constraints. While this feature is dependent on the selected dynamics model, the addition of inequality constraints would be particularly helpful at setting hardware bounds in the factor graph. For example, a control policy which regulates the throughput of each node could be bounded by a minimum throughput (i.e. 0bps) and some maximum throughput specific to the Rajant DX2-50 nodes. This work could be a natural extension to the Equality-Constrained LQR work from [20].

Finally, one last major future work consideration is a stability analysis of the controller design. Given the nature of wireless mesh networks, latency, packet loss, and failure of communication nodes are phenomena to be expected within the dynamics model. Consequently, a stability analysis to see the performance of the WMNLQR and DWMNLQR controllers when exposed to such phenomena would further illustrate the functionality of this work for a real world application.

Bibliography

[1] R. Bellman and S. Dreyfus. *Applied Dynamic Programming*. Princeton University Press, 1962.

[2] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[3] Dr. Timothy Chung. *Darpa Subterranean (SubT) Challenge*. `https://www.darpa.mil/program/darpa-subterranean-challenge`.

[4] Frank Dellaert and Michael Kaess. "Factor Graphs For Robot Perception". In: *Foundations and Trends in Robotics* 6.1-2 (2017), pp. 1–139. DOI: `10.1561/2300000043`.

[5] Marin Kobilarov Duy-Nguyen Ta and Frank Dellaert. "A Factor Graph Approach To Estimation and Model Predictive Control on Unmanned Aerial Vehicles". In: *ICUAS* 2014.27-30 (2014), pp. 181–188. DOI: `10.1109/ICUAS.2014.6842254`.

[6] Yetong Zhang Gerry Chen and Frank Dellaert. *LQR Control Using Factor Graphs*. `https://gtsam.org/2019/11/07/lqr-control.html`.

[7] Markus Giftthaler et al. "The Control Toolbox - An Open-Source C++ Library for Robotics, Optimal and Model Predictive Control". In: *2018 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*. May 2018, pp. 123–129. DOI: `10.1109/SIMPAR.2018.8376281`.

[8] Bob Givan and Ron Parr. *An Introduction To Markov Decision Processes*. `https://www.cs.rice.edu/~vardi/dag01/givan1.pdf`.

[9] Kevin Jamieson. *Lecture 20: Linear Dynamics and LQG*. `https://courses.cs.washington.edu/courses/cse599i/18wi/resources/lecture20/lecture20.pdf`. 2018.

[10] M. Kaess et al. "isam2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering". In: *IEEE International Conference on Robotics and Automation* (2011), pp. 3281–3288.

[11] H.J. Kappen, V. Gomez, and M. Opper. "Optimal control as a graphical model inference problem". In: (2009).

[12] Timo Koski and John M. Noble. *Bayesian Networks: An Introduction*. Wiley, 2009.

[13] Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. “Factor Graphs and the Sum-Product Algorithm”. In: *IEEE Transactions on Information Theory* (1998).

[14] *Linear Quadratic Regulator: Discrete-time finite horizon.* `https://stanford.edu/class/ee363/lectures/allslides.pdf`. 2009.

[15] P.Sodhi et al. “Ics: Incremental constrained smoothing for state estimation”. In: *IEEE International Conference on Robotics and Automation (ICRA)* (2020), pp. 279–285.

[16] Robert Platt. *Introduction to Linear Quadratic Regulation.* `http://www.ccs.neu.edu/home/rplatt/cs5335_2015/slides/lqr_writeup.pdf`. 2013.

[17] Roshan Pradhan et al. “Optimal Control for Structurally Sparse Systems using Graphical Inference”. In: *CoRR* abs/2104.02945 (2021).

[18] *Rajant.* `rajant.com`.

[19] Mandy Xie, Alejandro Escontrela, and Frank Dellaert. “A factor-graph approach for optimization problems with dynamics constraints”. In: (2020).

[20] Shuo Yang et al. “Equality Constrained Linear Optimal Control With Factor Graphs”. In: *arXiv* 1 (2020).