Anirudh Chari and Atharva Gawde

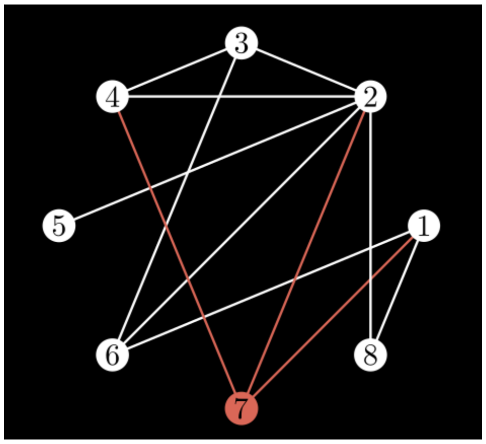
Mr. Meyer

Advanced Programming

22 May 2022

Applications of Graph Theory in Robotics: Riemannian Motion Policies

I. Introduction



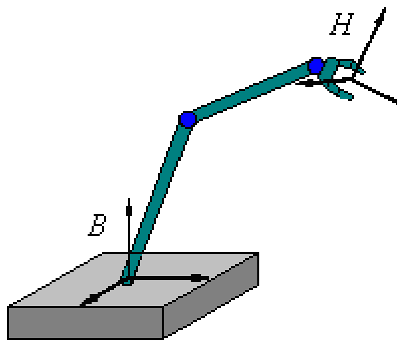
Graphs are widely used representations of both natural and man-made structures. They may be used to simulate a wide range of relationships and process dynamics in computer science, physical, biological, and social systems. Graphs can be used to illustrate a wide range of real situations. In general, graph theory has several applications in a variety of domains. Given its extensive applications, we focus on graph applications in computer science. This paper investigates the applications of graph theory within the field of robotics through Riemannian Motion Policies.

The Riemannian Motion Policy (RMP), first proposed in 2018 by researchers at Nvidia, is a second-order dynamical motion generation system, meaning it takes as input a robot's current position and velocity and outputs acceleration. Mathematically, this can be denoted as $f : x, \dot{x} \rightarrow \ddot{x}$. What distinguishes RMPs from other motion generation systems, however, is their ability to seamlessly integrate and dynamically prioritize various potentially conflicting tasks within a system. This paper explores how Riemannian Motion Policies function both individually and in conjunction with each other and with their environment, and the applications of this cutting-edge motion control system (Ratliff et al.).

II. Background

Reimannian Motion Policies are relatively math-heavy, so it is important to understand various foundational concepts. This section will give an overview of those concepts.

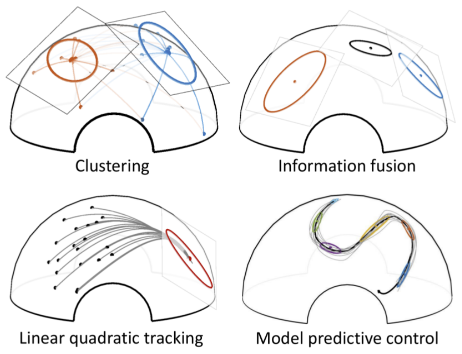
For a robot traveling in a two-dimensional space, its movement can be modeled using the Cartesian coordinate system (i.e. the conventional x-y plane). We can call this plane the configuration space, representing its state with the notation (q_0, q_1). However, if the robot needs to perform some task using its arm, it would be difficult to model the movement of that arm using (q_0, q_1). On the other hand, the arm cannot use a completely unrelated coordinate system, since its motion computations may depend on the motion of the drivebase, and vice versa. The solution to this problem is the linear algebra concept of linear transformation, which is some standardized function allowing, essentially, the entire configuration space plane to be shifted to temporarily align with the arm so it can easily complete its computations. In order to go back to the drivebase's coordinate system, the linear transformation is simply reversed. The newly generated coordinate system is referred to as the arm's task space, with task space coordinates denoted $(x_0,\ x_1)$, and the linear transformation used to generate the task space is called its task map (Ratliff et al. 2018). The figure to the left illustrates this transformation for configuration space B and task space H.



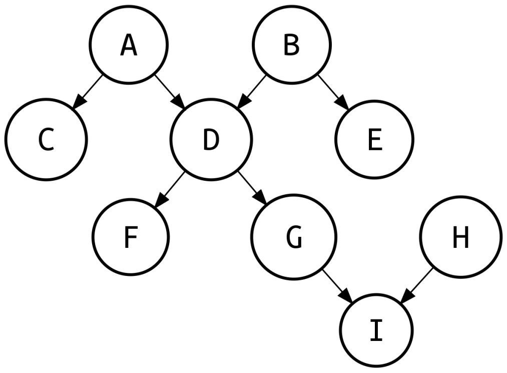
Next, one should understand the components of a single task's policy. A single RMP takes as input position and velocity $(x,\ \dot{x})$ in its own task space. The output of the policy is inspired by the Newtonian physics equation $F = ma$. The "force" outputted by the policy is the product of its desired acceleration $\ddot{x}$ in its own task space and its computed inertia matrix M, which is the relative importance of the calculated acceleration. The inertia matrix is the key to the flagship feature of Riemannian Motion Policies: dynamic task prioritization (Cheng et al. 2019).

Riemannian topology and geometry also allow more computationally elegant and efficient solutions to the problems put forth in autonomous navigation. Riemannian manifolds are relevant to a wide range of machine learning and robotics challenges. Primarily, Gaussian distribution-based robot learning applications have shown promise. This comprises procedures that necessitate uncertainty and statistical modeling on structured non-Euclidean data. One intriguing use of Riemannian geometry in robotics is that it provides a logical and straightforward approach to adapt algorithms originally designed for Euclidean data to other manifolds by efficiently taking pre-existing geometric information about these manifolds into account (Calinon).

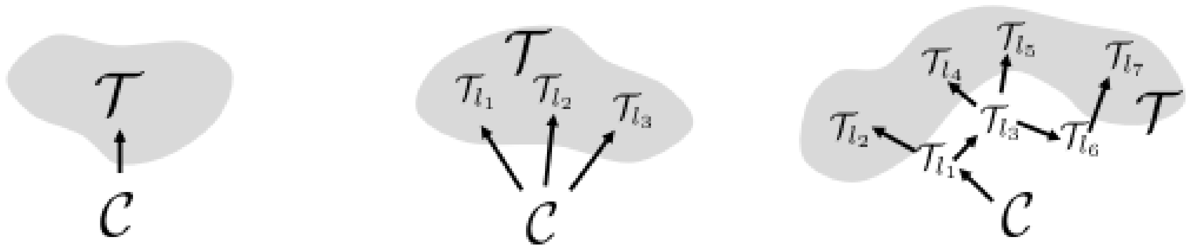
Data of multiple forms can be processed in a unified manner by employing Riemannian manifolds, with the benefit that current models and methods established for Euclidean data can be expanded to a larger range of data formats. It may be used to revisit restricted optimization problems defined in Euclidean space, for example, by considering them as unconstrained problems that take the geometry of the data into account (Calinon). Essentially mapping data onto geometry allows us to understand the information via the lens of pre-existing mathematical problems and tools.



One highly effective and typical strategy for studying graph theory and its applications is to focus on certain types of graphs. In the case of RMPs, directed trees/polytrees are the most beneficial. The difference between directed and undirected graphs in graph theory is that a directed graph has an ordered pair of vertices, whereas an undirected graph has an unordered pair of vertices. In layman's words, an edge can only be traversed in one direction. A tree is an undirected graph in which any two vertices are connected by exactly one route; similarly, a tree is a connected acyclic graph. In other terms, a tree is a connected graph with no cycles, having branches at the ends. Polytrees are thus, intuitively, trees with directed connections (John Adrian Bondy and USR Murty). It is commonly assumed in computer science that the trees being dealt with are rooted and oriented towards the branches at all vertices.



Thus we arrive at the data structure that puts everything together: the RMP tree, a polytree. The following are some examples of RMP trees, where C is the configuration space and T is the set of task spaces:



The configuration space is the root node, while tasks are leaf nodes (or have subtasks which are leaf nodes). The flow of information within an RMP tree at each timestep is a multi-step process. First, the configuration node receives the latest position and velocity data

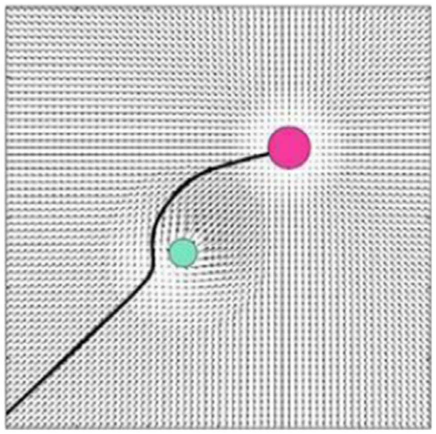
$(x,\ \dot{x})$ as input and propagates this data down to each task node through what is known as a pushforward method, which performs the corresponding task map to transform between coordinate systems. Next, each task node computes its desired acceleration matrix in its own task space, as well as that acceleration's inertia matrix. Then, this data from each task node is propagated back upward to the configuration node through what is known as a pullback method, which reverses the task map to obtain configuration-space acceleration matrices. Finally, the configuration combines all received information in order to generate an overall acceleration $\ddot{x}$ as output (Li and Mukadam 2019).

III. Examples

In order to exemplify Riemannian Motion Policies, two policies were implemented in Java using the RMPFlow framework. This section discusses those policies.

The first is a collision avoidance RMP (Ratliff et al. 2018), which takes as input upon initialization the Cartesian position (h, k) and radius r of some obstacle to avoid. The task map to the one-dimensional task space of the policy is given by the shortest Euclidean distance between the robot and the obstacle:

$x = \left|\sqrt{(q_0 - h)^2 + (q_1 - k)^2} - r\right|$. Intuitively, the behavior of the policy can be viewed as a sort of repelling magnetic field centered at the obstacle and curving the robot's path outward as it approaches, as depicted in the image on the right. The policy's inertia matrix is given by $M = \frac{\epsilon + min(0, \dot{x})\dot{x}}{x^4}$, where ϵ is some small positive tuning

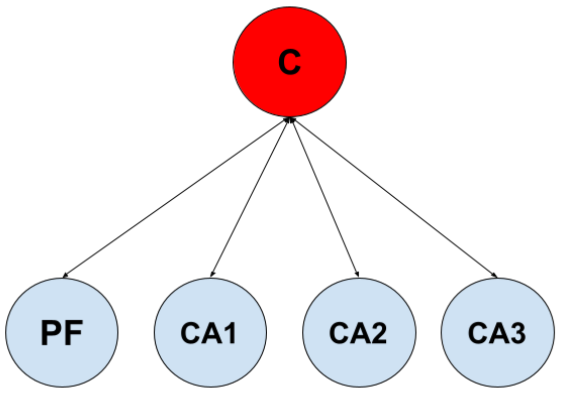


constant. This defines that the relative importance of avoiding a collision should depend on both the distance from the obstacle and the velocity towards the obstacle: decreasing distance or increasing velocity will drastically increase the relevance, and vice versa, with motion away from the obstacle (i.e. negative task space velocity) putting the node in a temporary "sleep" state. (Li and Mukadam 2019). The acceleration matrix, then, is computed by $\ddot{x} = \frac{\eta}{2x^4}$, where η is a tuning constant. In order to visualize this, imagine tying a string between the robot's start and goal points, then pulling this string taut across any obstacles between the two points. The resulting shape of the string is the robot's path, with the shape and size of the deformation in the string due to the obstacle being determined by the acceleration and inertia matrices, respectively.

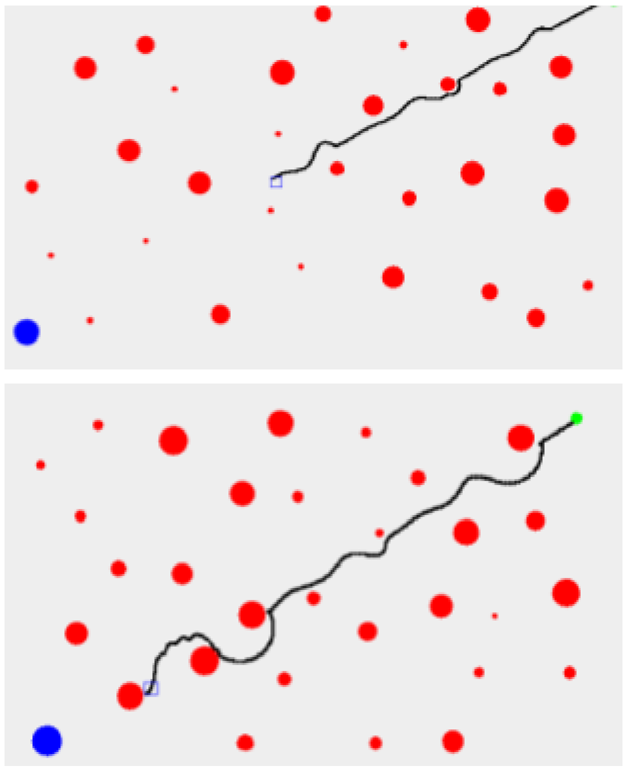
The second policy implemented is a path following RMP, which takes as input upon initialization some path in the Cartesian plane. For simplicity, the path given to the policy in all trials throughout this paper was a straight line between the start and end points, but paths generated by planning algorithms can be passed as input as needed. While the collision avoidance policy was designed in the publication introducing RMPs, the path following policy was designed and proven entirely by IMSA's robotics team. The task space is two-dimensional, with one dimension being the robot's distance along the path, and the other being its perpendicular distance from the path. The abbreviated computations for this task map, assuming a linear path, are $x_0 = dist(q,\ s) \times cos(angle(e,\ s,\ q))$ and $x_1 = dist(q,\ pap(x_0))$, where $dist(a,\ b)$ returns the distance between points a and b, s is the start point of the path, $angle(a,\ b,\ c)$ returns the angle between three points given base a, vertex b and leg c, e is the end point of the path, and $pap(d)$ returns the position along the path after some distance d. Task mapping for nonlinear paths involves polynomial approximation and other calculus techniques, which is beyond the scope of this paper. The inertia matrix is given by $M = [ksin(\frac{\pi}{2}h), kcos(\frac{\pi}{2}h)]$, where k and h are tuning constants. This means that the relative importance of the path following task can be configured upon initialization, but it stays constant at those values throughout the path. Finally, the acceleration matrix can be mathematically represented as $\ddot{x} = [P(v - \dot{x}_0) + I\varepsilon, Ax_1 - B\dot{x}_1]$, where P, I, A, and B are tuning constants resembling a closed-loop control system, v is some some desired velocity, and ε is the accumulated error in observed velocity versus desired velocity. Desired velocity is computed by $v = min(v_{targ}, \sqrt{2a_{max}d})$, where v_{targ} and a_{max} are, respectively, the target velocity and maximum acceleration configured upon the policy's initialization, and d is the current distance along the path. This computation enables the robot to smoothly speed up to target velocity upon starting, and smoothly slow down to rest as it approaches the goal. Putting everything together, the acceleration generated by the path following RMP can be intuitively visualized as an imaginary spring connecting the robot to the point along the path that it should be at in the next timestep. The spring generates force both along and towards the path, with greater displacement leading to greater magnitude of force, and the spring constant (i.e. stiffness) is determined by the inertia matrix.

IV. Demonstration

In order to demonstrate Riemannian Motion Policies, a mobile robot simulation was developed in Java using the RMPFlow framework. Each instance of the simulation generates a randomized environment ridden with n obstacles and arbitrary robot start and goal positions. Consequently, the robot is assigned $n + 1$ tasks: one for path following, and one for collision avoidance of each of the n obstacles. The figure to the right illustrates the RMP tree corresponding to an environment with three obstacles, with the root



node C representing the configuration space, the leaf node PF representing the path following task space, and each leaf node $CA\{i\}$ representing each obstacle's collision avoidance task space. The tree's edges, all of which are bidirectional, represent the downward propagation of position and velocity data and the upward propagation of acceleration data. Each edge also represents a task map between the configuration space and a task space. Edge weights can potentially be interpreted as dynamic, since the relative importance (i.e. inertia) of each policy changes at each timestep, although inertia data is propagated upward along with acceleration data. Pictured to the right are two unique instances of the simulation, rendered using Java's Swing library. For reference, the blue square is the robot, the blue circle is the goal position, the red circles are 30 obstacles of varying sizes, and the black line is the traversed path. Target velocity was configured to 8.0 m/s and maximum acceleration to ± 2.0 m/s^2 based on the constraints of a standard competition robot, and adjustment of the tuning constants of each policy was made based on trial and error. Visible demonstration in the pictured instances of successful prioritization and integration among tasks is that when the robot approaches an obstacle, it always circumvents the obstacle in the direction that displaces it the least from the path. Observation of the simulation through dozens of randomized trials proved that RMPs generated safe, fast, and efficient motion through successful dynamic prioritization of various tasks. Additionally, the algorithm's fluid adaptations to the



environment provided the robot with excellent maneuverability: across a simulated ~20 km^2 field ridden with 30 obstacles, a 100 cm^2 robot only required ~50 cm of distance between the edges of adjacent obstacles within a group in order to successfully maneuver through that group. When this lower bound was programmed into the simulation's obstacle generation algorithm, the robot was able to generate coherent, visibly optimal motion toward the goal, reaching the goal safely and briskly in every trial performed.

V. Applications

Graph Theory is ultimately the study of relationships. Given a set of nodes and connections, graphs can abstract anything from city layouts to computer data, graph theory provides a helpful tool to quantify and simplify the many moving parts of dynamic systems. Studying graphs through a framework provides answers to many arrangement, networking, optimization, matching and operational problems (Foulds). Graphs can be used to model many types of relations and processes in physical, biological, social and information systems, and has a wide range of useful applications such as finding communities in networks, such as social media, Ranking/ordering hyperlinks in search engines, GPS/Google maps to find the shortest path home, Study of molecules and atoms in chemistry, DNA sequencing, and many more (Flovik).

Humanoid robots, unmanned rovers, entertainment pets, drones, and other mobile robots are prime examples of the many use cases of autonomous navigation systems and Riemannian Motion Policies. They differ from other robots in that they need to move autonomously and have enough intelligence to respond and make decisions based on their assessment of their surroundings. To adapt to a changing reality, mobile robots must have a source of input data, a method of decoding that input, and a method of performing actions (including their own

mobility). The requirement to detect and adapt to an unfamiliar environment necessitates a sophisticated cognitive system, where Riemannian Motion Policies come into play.
Several domains of robotics have emerged, such as wheeled mobile robots, legged robots, flying robots, robot vision, artificial intelligence, and so on, including several technological areas such as mechanics, electronics, and computer science (Majeed and Rauf). These emerging developments are headed by artificial intelligence, autonomous driving, network communication, nanorobotics, pleasant human-robot interfaces, safe human-robot interaction, and emotional expression and perception. Furthermore, these news developments are utilized in several industries such as medical, health care, sports, ergonomics, industry, product distribution, and service robots, a trend that will continue to evolve in the future years.

Mobile robotics is currently one of the most rapidly increasing disciplines of scientific inquiry. Because of their capabilities, mobile robots can replace humans in a variety of industries. Surveillance, planetary exploration, patrolling, emergency rescue operations, reconnaissance, petrochemical applications, industrial automation, construction, entertainment, museum guides, personal services, intervention in extreme environments, transportation, medical care, and many other industrial and nonindustrial applications are examples of applications. Thus, the research and development of dynamical systems and motion policy are imperative to improve the abilities and efficiency of tools in this growing field. The basics of mobile robotics consist of the fields of locomotion, perception, cognition, and navigation, and Riemannian Motion Policies and their foundation in the mathematics of Riemannian Manifolds provide insight into approaching and conquering these four fields cooperatively (Wingo et al.).

Riemannian Motion Policies are a combined representation of the robot's and its environment's states. It simulates the interaction between a point on the robot and its surroundings using an acceleration strategy and a Riemannian metric. The best control instructions may be calculated analytically by utilizing the kinematic model. In other words, it combines robot geometry, dynamics, and local impediments into a single representation, addressing locomotion, perception, cognition, and navigation all simultaneously.

These RMP structures can also be incorporated into the architecture of a neural autonomous navigation system. When compared to networks taught to anticipate depth or control instructions, neural networks may be trained in parallel to predict RMPs and generalize to previously explored or easily computable environments. The behavior of the vehicle may be accurately reasoned by studying the expected RMPs, resulting in more explainable neural vehicle control. *Vision-based vehicle control and navigation* and *Model-based robot control* are paired to create more comprehensive navigation models.

Works Cited

Biggs, Norman L., et al. *Graph Theory 1736-1936*. Clarendon Press, Dr, 2006.

Cheng, Ching-An, et al. "RMPflow: A Computational Graph for Automatic Motion Policy Generation." *ArXiv:1811.07049 [Cs]*, Apr. 2019, arxiv.org/abs/1811.07049.

Deo, N. "GRAPH THEORY with APPLICATIONS to ENGINEERING and COMPUTER SCIENCE." *Networks*, vol. 5, no. 3, July 1975, pp. 299–300, https://doi.org/10.1002/net.1975.5.3.299.

Flovik, Vegard. "What Is Graph Theory, and Why Should You Care?" *Medium*, 19 Aug. 2020, towardsdatascience.com/what-is-graph-theory-and-why-should-you-care-28d6a715a5c2.

Foulds, L. R. *Graph Theory Applications*. Springer-Verlag, 1992.

Gross, Jonathan L., et al. *Handbook of Graph Theory*. Chapman & Hall/Crc, 2017.

John Adrian Bondy, and U S R Murty. *Graph Theory with Applications.* London Macmillan, 1977.

Li, Anqi, and Mustafa Mukadam. "Multi-Objective Policy Generation for Multi-Robot Systems Using ..." *Multi-Objective Policy Generation for Multi-Robot Systems Using Riemannian Motion Policies*, 14 Feb. 2019, https://www.researchgate.net/publication/331111152_Multi-Objective_Policy_Generation_for_Multi-Robot_Systems_Using_Riemannian_Motion_Policies.

Majeed, Abdul, and Ibtisam Rauf. "Graph Theory: A Comprehensive Survey about Graph Theory Applications in Computer Science and Social Networks." *Inventions*, vol. 5, no. 1, Feb. 2020, p. 10, https://doi.org/10.3390/inventions5010010.

PalSingh, Rishi, and Vandana Vandana. “Application of Graph Theory in Computer Science and Engineering.” *International Journal of Computer Applications*, vol. 104, no. 1, Oct. 2014, pp. 10–13, https://doi.org/10.5120/18165-9025.

Ratliff, Nathan D., et al. “Riemannian Motion Policies.” *ArXiv:1801.02854 [Cs]*, July 2018, arxiv.org/abs/1801.02854v3.

“Research at NVIDIA: RMPflow - a Computational Graph for Automatic Motion Policy Generation.” *Www.youtube.com*, www.youtube.com/watch?v=Fl4WvsXQDzo. Accessed 19 May 2022.

Riaz, Ferozuddin, and Khidir M. Ali. “Applications of Graph Theory in Computer Science.” *IEEE Xplore*, 1 July 2011, pp. 142–45, https://doi.org/10.1109/CICSyN.2011.40.

Wingo, Bruce, et al. “Adaptively Robust Control Policy Synthesis through Riemannian Motion Policies.” *IEEE Control Systems Letters*, vol. 6, 2022, pp. 31–36, https://doi.org/10.1109/lcsys.2020.3047359.