

12-12-2016

An Algorithm for Calculating the Inverse Jacobian of Multirobot Systems in a Cluster Space Formulation

Christopher Jude Waight
Santa Clara University

Follow this and additional works at: http://scholarcommons.scu.edu/mech_mstr



Part of the [Mechanical Engineering Commons](#)

Recommended Citation

Waight, Christopher Jude, "An Algorithm for Calculating the Inverse Jacobian of Multirobot Systems in a Cluster Space Formulation" (2016). *Mechanical Engineering Master's Theses*. 7.
http://scholarcommons.scu.edu/mech_mstr/7

This Thesis is brought to you for free and open access by the Engineering Master's Theses at Scholar Commons. It has been accepted for inclusion in Mechanical Engineering Master's Theses by an authorized administrator of Scholar Commons. For more information, please contact rscroggin@scu.edu.

Santa Clara University
DEPARTMENT of MECHANICAL ENGINEERING

Date: December 12, 2016

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY
SUPERVISION BY

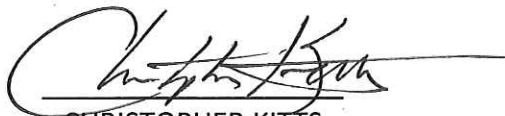
Christopher Jude Waight

ENTITLED

**An Algorithm for Calculating the Inverse Jacobian of Multirobot Systems
in a Cluster Space Formulation**

BE ACCEPTED IN PARTIAL FULLFILMENT OF THE REQUIREMENTS FOR THE
DEGREE OF

MASTER OF SCIENCE IN MECHANICAL ENGINEERING



CHRISTOPHER KITTS
THESIS ADVISOR



TERRY SHOUP
READER



DRAZEN FABRIS
DEPARTMENT CHAIR

*An Algorithm for Calculating the Inverse Jacobian of
Multirobot Systems in a Cluster Space Formulation*

by

Christopher Jude Waight

MASTER OF SCIENCE THESIS

Submitted in partial fulfillment of the requirements
for the degree of
Master of Science in Mechanical Engineering
School of Engineering
Santa Clara University

Santa Clara, California
December 12, 2016

An Algorithm for Calculating the Inverse Jacobian of Multirobot Systems in a Cluster Space Formulation

Christopher Jude Waight

Department of Mechanical Engineering
Santa Clara University
2016

Abstract

Multirobot systems have characteristics such as high formation re-configurability that allow them to perform dynamic tasks that require real time formation control. These tasks include gradient sensing, object manipulation, and advanced field exploration. In such instances, the Cluster Space Control approach is attractive as it is both intuitive and allows for full degree of freedom control. Cluster Space Control achieves this by redefining a collection of robots as a single geometric entity called a cluster. To implement, it requires knowing the inverse Jacobian of the robotic system for use in the main control loop. Historically, the inverse Jacobian has been computed by hand which is an arduous process. However, a set of frame propagation equations that generate both the inverse position kinematics and inverse Jacobian has recently been developed. These equations have been used to manually compile the inverse Jacobian Matrix. The objective of this thesis was to automate this overall process. To do this, a formal method for representing cluster space implementations using graph theory was developed. This new graphical representation was used to develop an algorithm that computes the new frame propagation equations. This algorithm was then implemented in Matlab and the algorithm and its associated functions were organized into a Matlab toolbox. A collection of several cluster definitions were developed to test the algorithm, and the results were verified by comparing to a derivation based technique. The result is the initial version of a Matlab Toolbox that successfully automates the computation of the inverse Jacobian Matrix for a cluster of robots.

Keywords: Multirobot, Cluster Control, Inverse Jacobian, Propagation Equations, Algorithm

Acknowledgements

I would like to thank my advisor, Dr. Chris Kitts, for suggesting exploring this technique, and for providing the framework needed for this research. I would also like to thank him for his encouragement, patience, and perspectives on several issues that were encountered throughout this research. I would also like to thank him for helping me to understand what my contribution to the field is.

I would also like to thank everyone in the SCU Robotics Lab. Thanks goes to former students, as much of their previous research was used as references for my understandings. Thanks goes to current students who are also working on Cluster Control, and offering insights and explanations to some of the finer details of the control method. Thanks goes to Alex Mulcahy for providing a test case for the algorithm that was developed. Thanks goes to all others too, for the encouragement, and for the company through the long hours of work.

I would like to thank my family for their support and enthusiasm as I pursued this degree. Thanks mom and dad for all the encouraging words, and to my brothers and cousins for understanding the business of my schedule. I would also like to thank Irene and Harry Partridge for all their help, encouragement, support, and for being awesome people. Without them, none of this would be possible.

A final thank you to all my close friends. I will never forget any of the phone calls, the care packages with food and encouraging notes, the music recommendations, and for reminding me to go big, or to go home.

Table of Contents

Abstract	iii
Acknowledgements	iv
List of Figures	vii
List of Tables.....	viii
Chapter 1: Introduction.....	1
1.1 Formation Control of Multirobot Systems	1
1.2 Cluster Space Control	2
1.3 Inverse Jacobian Matrices	4
1.4 Project Statement	6
1.5 Readers Guide	6
Chapter 2: Graph Based Representations of Clusters	7
2.1 Determining Suitable Cluster Space Variables	7
2.2 The Homogeneous Transform in terms of Cluster Space	8
2.3: Graph Based Representations of Clusters	9
2.4 Determining the Inverse Kinematics from the Graph Based Representations.....	11
2.5 A few notes on the Tree Representation of a cluster space formulation	12
Chapter 3: Jacobian Propagation Algorithm	14
3.1 The Inverse Jacobian Algorithm	14
3.2 An implementation of the Formula.....	16
3.3 Run Time Analysis of this algorithm	21
Chapter 4: Matlab Implementation of Algorithm	23
4.1 The Cluster_Builder Folder	24
4.2 The Inverse Kinematics Folder	27
4.3 The Test_Bed Folder.....	28
4.4 The Velocity_Propagation_Technique Folder	31
Chapter 5: Results	33
5.1 Results from Test_Bed Examples	33
5.2 Results from a Third Party Cluster.....	33
5.3 Further Testing II	34
Chapter 6: Conclusions.....	36
6.1 Summary	36
6.2 Future Work	37

References.....	38
Appendix A – Examples used in the Test_Bed Folder	40
Appendix B – Code used in Toolbox	62
B.1 Tree Data Structure Folder.....	62
B.2 Cluster_Builder Folder.....	62
B.3 The Inverse_Kinematics Folder	67
B.4 The Velocity_Propagation_Technique Folder	70
Appendix C - The Inverse Jacobian Formula: A Proof.....	79

List of Figures

Figure 1: A Resolved Rate Cluster space control architecture for a generic multirobot system	3
Figure 2: A Cluster of 3 robots, with the distance variables M, N, and L shown.....	8
Figure 3: A Physical View of Example3, a cluster of two 2-robot clusters.....	9
Figure 4: A group of unconnected vertices representing all the frames in a cluster.....	9
Figure 5: Two 2-Robot Clusters represented by the set of edges E1.	10
Figure 6: A 2 robot cluster in a leader follower relationship with another 2 robot cluster and a fifth robot in a leader follower relationship with cluster frame C2	10
Figure 7: A completed Cluster Graph of a 5 Robot Cluster	11
Figure 8: A Robot with two leaders	13
Figure 9: Initial tree with all nodes colored white	17
Figure 10: Tree with current node colored yellow	17
Figure 11: Cluster tree with explored node colored red and current node yellow	18
Figure 12: Cluster tree with all nodes from leaf to root explored once.	18
Figure 13: Returning to the robot node of the current subroutine	18
Figure 14: Cluster Tree with Explored nodes are green and current node is yellow	19
Figure 15: Returning to robot node being computed and mark as current node	20
Figure 16: A cluster tree with a single fully explored robot node	20
Figure 17: A fully explored robot node and a new robot node selected	20
Figure 18: A fully explored cluster tree	21
Figure 19: Directory of the Cluster Space Control Inverse Jacobian Toolbox	23
Figure 20: User entering tree information for example3 into clusterbuilder2.m	25
Figure 21: User entering HTM matrices for each edge on the tree of example3	25
Figure 22: Display of attributes initially saved in the Matlab variable example3	26
Figure 23: The function invkin.m being used on example3 variable	27
Figure 24: The function dirinvjac.m being used on example3 variable	28
Figure 25: A Physical View of Example3, a cluster of two 2-robot clusters	28
Figure 26: A Graph representing a cluster of clusters. Cluster {C1} and {C2} form a single cluster {C}	29
Figure 27: The function invjacfxn.m being used on example3 variable	32
Figure 28: The Alex Cluster. A 5 robot cluster with robots 1 and 2 forming the main cluster, robot 4 following robot 2, robot 3 following robot 1, and robot 5 following robot 3	33
Figure 29: Columns 1 through 7 of the inverse Jacobian of the Alex Cluster	34
Figure 30: Columns 8 through 15 of the inverse Jacobian of the Alex Cluster	34

List of Tables

Table 1: A list of all the files in the Cluster_Builder folder.....	24
Table 2: A list of all the variable attributes saved in each example variable	26
Table 3: Description of all Files in the Inverse Kinematics folder	27
Table 4: Examples in the test bed folder and an explanation of their configuration	31
Table 5: Descriptions of the files in the Velocity_Propagation_Technique Folder	32
Table 6: Other Cluster Definition to test Robustness of Toolbox	35

Chapter 1: Introduction

Robots have been instrumental in developing many industries in modern culture. This is because a robot can be developed to complete tasks faster, with higher precision, and more cost effectively than human labor. It can work in harsh environments, complete repetitive tasks with less wear and tear than humans, and can perform tasks that are far more complicated. As the tasks that a robot can perform grow in complexity, so does the task of developing a system to control the robot. This challenge in developing a system of control grows even more complicated when multiple robots need to be controlled at the same time. This has created the current demand for a control approach that is intuitive, reliable, and most importantly, scalable.

1.1 Formation Control of Multirobot Systems

A multirobot system (MRS) can be defined as a set of robots operating in the same environment in an independent, cooperative, or competitive manner. The term robot applies to any electro-mechanical agent that is guided by a computer program or electronic circuitry. This definition allows the MRS scope to range from a group of two sensor equipped actuators to a large set of complex humanoid machines with hundreds of sensors and actuators that interact with the environment and each other using very complex decision making [1].

An MRS offers many advantages when compared to a single robot system. Having multiple robots performing the same task can increase coverage, production scales, and redundancy. Other advantages include increasing configurability, more modularity, and the ability to share sensor information. Some benefits of a MRS can only be achieved as a result of robots working in a cooperative fashion to complete a task that an individual robot cannot complete on its own. For example, a robot can hand off a task that it is unable to complete to another robot in the system. Robots can even work through coordinated and cooperative behaviors to accomplish tasks such as manipulating large unbalanced objects through obstacle filled paths. These advantages can be realized on land based agents, and also for applications in air, sea, and space [2].

Systems of multiple robots have their origins in the late-1980s, and, to date, have been applied in several domains that require complex co-ordination. For example, the CENTIBOTS project created an experimental demonstration showing a large team of robots (approximately 100) that could autonomously patrol a building for an extended period of time. [4] In the FIRE project, a simulation of a team of intelligent heterogeneous robots that explored harsh, humanly inaccessible planetary surfaces was created [5]. Other projects include simulations of satellite formations [6], a test bed of underwater robot fleets [7], and a test bed of unmanned aerial robots [8]. Implementation of MRS's is still in its infancy, with none of the mentioned examples operating routinely due to ongoing technical challenges. Nonetheless, the study of MRS's is a growing field of interest especially as technological improvements in both hardware and software are developing exponentially [3].

The control of robotic systems is a matter of ongoing exploration and draws on work in control theory, robotics, biology, and artificial intelligence. The control strategy employed is dependent on the classification of the MRS based on the level of awareness, coordination, cooperation, and information sharing required [1]. Sometimes, it is possible to control a system of robots by controlling the behavior of each robot independently. More often, however, systems need to be strongly coordinated, and the

system needs to combine sensor information across multiple robots to make an informed decision of how the system as a whole should be controlled. Several control techniques have been proposed, each with its own pros and cons.

One version of control includes the use of leader follower techniques, in which ‘follower’ robots regulate their position relative to a designated ‘leader’ robot [9] [10]. This approach requires diligence on part of the operator defining the system, as two robots with similar ‘follow’ instructions can compete to occupy the same space and result in collision. A variation of this technique is to conduct leader-follower chains. In a leader follower chain, one robot may be following a ‘leader’ robot, while simultaneously acting as a leader for another robot [11]. These techniques are highly susceptible to propagation error as errors accumulate down the follower chains. Furthermore, the failure of a single robot can cause the entire chain to no longer function.

Another technique used for formation control is the creation of artificial potential fields. These fields can be used to attract individual robots to their desired formation location, while others are used to repel them from nearby robots or other obstacles. In this approach, a robot (or robots) can emit some signal that assigns some ‘penalty’ to other robots nearby based on their position. The robots receiving this penalty have some heuristic (usually to minimize penalty) and can then change its own location to minimize this penalty. Similarly, the robot might have a heuristic to get equal penalties from two or more other robots in this system. This technique is widely used in obstacle avoidance algorithms, as it allows robots to arrange themselves in formations that prevents them from colliding with each other or some other obstacle [12] [13] [14] [15]. These methods allow for fast computation, but do not offer a high level of controllability.

Multirobot control has also been a growing interest in the field of artificial intelligence. Algorithms considering multiagent systems composed of multiple interacting intelligent agents being developed and simulated for possible use in search and rescue, transportation, and reconnaissance [16] [17]. These techniques tend to be very computationally expensive, but work is being done to improve the computational costs [19].

Cluster space control, a control methodology developed and tested at the Santa Clara University Robotic Systems Lab offers an approach that is intuitive, stable, and scalable. It also allows a full degree of freedom to be maintained, and results in very precise control over individual robot control. This strategy conceptualizes an ‘n’ robot system as a single entity called a cluster. The desired positions and motions of the individual robots are then specified as a function of cluster states [18]. This method comes at the cost that the controller can be quite complex to develop. This research aims to make developing new cluster space controllers easier.

1.2 Cluster Space Control

Cluster Space Control is a control methodology that allows for the specification, control and monitoring of the motion a MRS [14]. This strategy considers a system of n robots as a single entity, known as a cluster. This cluster is modeled as a virtual articulating mechanism with a full degree of freedom.

In this methodology, we first consider \vec{R} to be set of robot state pose variables. These variables describe the position and orientation of each robot relative to the global frame. We then consider a selection of cluster space variables, \vec{C} , which describe the position and orientation of overall cluster,

the shape of the cluster, and the orientation of individual robots within the cluster. This set of variables, \vec{C} , can be defined through a formal set of forward kinematic transforms of robot state variables, \vec{R} .

$$\vec{C} = \text{KIN}(\vec{R}) \quad (\text{Eq 1})$$

Similarly, we can use the inverse kinematic transforms to take us from cluster space pose variables to robot state variables.

$$\vec{R} = \text{INVKIN}(\vec{C}) \quad (\text{Eq 2})$$

In other words, we can control the positions, motions, and even the actuator states of each robot can be specified as a function of cluster state variables.

In order to map the velocities from Robot Space, \vec{R} , to Cluster Space, \vec{C} , the velocity kinematics of the system also need to be known. The velocity kinematics can be found by computing the partial derivatives of the kinematic equations, g_i , with respect to each robot variable, r_i .

$$\vec{\dot{C}} = \begin{bmatrix} \dot{c}_1 \\ \dot{c}_2 \\ \vdots \\ \dot{c}_q \end{bmatrix} = \text{KIN}(\dot{G}\vec{R}) = J(\vec{R})\vec{\dot{R}} = \begin{bmatrix} \frac{dg_1}{dr_1} & \dots & \frac{dg_1}{dr_f} \\ \vdots & & \vdots \\ \frac{dg_q}{dr_1} & \dots & \frac{dg_q}{dr_f} \end{bmatrix} \begin{bmatrix} \dot{r}_1 \\ \dot{r}_2 \\ \vdots \\ \dot{r}_f \end{bmatrix} \quad (\text{Eq 3})$$

This matrix of partial derivatives, $J(\vec{R})$, is known in Cluster Space as the Jacobian Matrix. It is also referred to as simply the Jacobian. Similarly, the inverse Jacobian, $J^{-1}(\vec{R})$, transforms Cluster Space velocities to robot space velocities. The Inverse Jacobian can be found by taking the partial derivatives of the inverse kinematics, h_i , with respect to each cluster space variable, c_i :

$$\vec{\dot{R}} = \begin{bmatrix} \dot{r}_1 \\ \dot{r}_2 \\ \vdots \\ \dot{r}_f \end{bmatrix} = \text{INVKIN}(\dot{G}\vec{C}) = J^{-1}(\vec{C})\vec{\dot{C}} = \begin{bmatrix} \frac{dh_1}{dc_1} & \dots & \frac{dh_1}{dc_q} \\ \vdots & & \vdots \\ \frac{dh_f}{dc_1} & \dots & \frac{dh_f}{dc_q} \end{bmatrix} \begin{bmatrix} \dot{c}_1 \\ \dot{c}_2 \\ \vdots \\ \dot{c}_q \end{bmatrix} \quad (\text{Eq 4})$$

Using KIN, J, and J^{-1} , a cluster space controller can be developed as shown in figure 1 below:

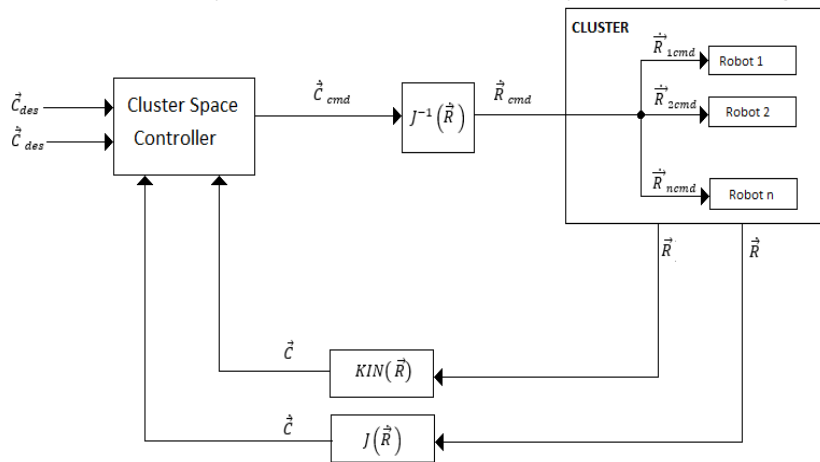


Figure 1: A Resolved Rate Cluster space control architecture for a generic multirobot system.

In this architecture in figure 1, the desired velocity control action is inputted in cluster space. The control actions are converted to robot space using the inverse Jacobian, and then passed to the individual robots. This example architecture also shows a feedback loop returning to the controller.

As figure 1 shows, determining the Jacobian and inverse Jacobian is essential to developing a cluster space controller [14] [18]. However, determining the inverse Jacobian can be quite difficult, and many approaches have been suggested over the years.

It is worth noting also that a different version of a controller for cluster space has been developed. This other version is a full dynamic controller and uses the transpose Jacobian as a replacement for the inverse Jacobian [30]. This controller requires further work to develop a systematic way to compute the forward Jacobian.

1.3 Inverse Jacobian Matrices

The need for finding the inverse Jacobian is not unique to developing cluster space controllers. When developing a controller for serial manipulators, the most common method involves using a Jacobian based Controller [20]. As a result, finding different efficient methods of determining the inverse Jacobian is essential for continued work in the field. Several methods for determining the inverse Jacobians exist with analytical methods generally being infeasible and computational methods commonly yielding less accurate results.

One technique for determining the inverse Jacobian is to solve for the inverse kinematics of the system, and then compute the partial derivatives. For a serial link manipulator, the closed form inverse kinematic solution can be obtained symbolically by writing a system of equations defining the forward kinematic relationships, and then solving the system of equations for the joint angles [20] [21]. However, this often cannot be solved in closed form. Geometric and trigonometric methods of determining an analytic form of inverse kinematics exist, but usually do not contain the full geometric description. Therefore, these methods provide solutions that are only valid in some local vicinity, and often allow for multiple solutions [21] [22] [24]. Other methods still are based on Denavit-Hartenberg, and rely on transformation matrices [23]. These approaches are also limited, as they restrict frame assignments throughout the entire robot. As a result, the inverse kinematics are typically solved numerically [25].

On the other hand, parallel-link manipulators often have a geometry that allow for closed form solutions to the inverse kinematics. Hence, the inverse Jacobian can be calculated by direct partial differentiation. A drawback in these systems is that the forward kinematics are nonlinear, and often require numerical approaches, and as a direct result, the forward Jacobian is solved numerically [26].

Instead of solving for the inverse Jacobian directly, another approach is to solve for the analytic form of the Jacobian, then invert it. The kinematics can be defined symbolically, then differentiated directly, as the Jacobian Matrix is a matrix of partial derivatives [27]. But for some systems, as seen with the case of the parallel link manipulators, calculating the symbolic form of the kinematics is not always possible. Furthermore, when this method is possible, it is not particularly efficient computationally, as it requires using a computational toolbox that has a symbolic differentiation library.

Several other methods for computing the Jacobian, particularly in serial manipulators, have been developed over the years. One method by Vukobratovic and Potkonjak recursively computes the Jacobian for each joint in the manipulator. It does this by making use of frame propagations. First, it computes the Jacobian for a manipulator with the first link only, then for the first two links, and so on and so forth up N links plus the end effector. More efficient methods based on the works of Pieper and Whitney utilize the property that rotational joint velocities add linearly, and translational velocities may be determined by taking appropriate cross-products of in the individual joint rate vectors and the position vector from that joint. Others have also used skew symmetric matrices to compute the Jacobian geometrically. These methods have been used widely throughout the field and their computational efficiencies have been compared to each other [28].

When the Jacobian cannot be found analytically, it is often found using a perturbation method using a first order numerical difference. A small change in each joint (or parameter) is made systematically, and the resulting movement is used to obtain an approximate numerical solution [25]. This method is only sufficient in the vicinity which the Jacobian was determined. Furthermore, difficulties arise when choosing the size of the small difference to use for the method. Too large a change will result in an inaccurate function if the systems dynamics are nonlinear. Too small a change will lead to numerical problems and greater inaccuracies [25].

While finding the Jacobian can be challenging on its own, several problems can arise when inverting the Jacobian Matrix. Typically, problems arise when the matrix is singular or has a poor condition number. Other problems arise when the Jacobian Matrix is not square. If the Jacobian is in numerical form but is not square, a Moore Penrose Inverse, or least squares inverse, can be used to compute the pseudo-inverse. In the event that the matrix is singular, or has a poor condition number, a damped least squares approach can be used. The damped least squares approach will not yield to an accurate result, but usually one that is close enough for engineering purposes. [29] These techniques to inverting the matrix only work numerically. The analytic form of the Jacobian matrix may be evaluated to make use of these numerical techniques.

Historically, determining both the kinematics and inverse kinematics of a system has been quite difficult. For this reason, some roboticists use a transpose Jacobian instead to create a dynamic controller instead of the Inverse Jacobian for use in a resolved rate controller [20]. This technique can be implemented both analytically and numerically. In a strictly Cartesian manipulator, the inverse of the Jacobian, J , is equal to the transpose of the Jacobian ($J^T = J^{-1}$). This is not true for other cases, but often a dynamic controller yields satisfactory results. However, poor performance has also been noted from this type of controller. Cluster space control, however, is a system that allows for systematic determination of both the forward kinematics and inverse kinematics, therefore, allowing for quick computation of both the forward and inverse Jacobian by computing the partial derivatives. This allows a resolved rate controller to be developed systematically.

This research will present a systematic method to compute the analytic inverse Jacobian for cluster space control systems given only the geometric description. Similarly, to the serial manipulators, frame propagation will provide the foundation to the approach. A collection of Matlab files has been created to compute this inverse Jacobian for a large variety of MRS systems. Creating an algorithm to implement this technique provides a quick and accurate way to determine the inverse Jacobian compared to the method of computing partial derivatives of the inverse kinematics. This is essential to further the study of cluster space control.

1.4 Project Statement

The purpose of this research is to develop a software suite to implement a new frame propagation technique for generating the inverse Jacobian matrix for cluster space formation of mobile robots. In carrying out this work, a significant amount of effort and interest has been directed toward the following tasks:

- 1) Developing a small library of Cluster Space Formulations as a test bed.
- 2) Formalizing a method for representing cluster space formulations using graph theory.
- 3) Using this graphical representation to develop an algorithm for implementing the new frame propagation technique.
- 4) Writing an implementation of the algorithm in Mathworks Matlab.
- 5) Writing additional Matlab files that compute the inverse Jacobian using a known technique.
- 6) Calculating the inverse Jacobian of all examples in the test bed using both techniques and comparing results against each other as a verification process.

The discovery and successful implementation of this new technique will allow for cluster space controllers to be developed more rapidly. Further work on this software suite can add features such as automatically developing an entire controller for a given cluster, a plot creator to compare velocities in different spaces, as well as adding more examples to the example folder.

1.5 Readers Guide

This section, Chapter 1, provides a quick introduction to the control of multirobot systems, challenges in developing these control methods, and possible drawbacks for each proposed method. This chapter also introduces the cluster control architecture and provides some background on current ways to determine the inverse Jacobian of a system.

Chapter 2 introduces the homogeneous transformation matrix, and shows how it can be used to represent a cluster as a graph. The conventions for creating such a graph are outlined, and a systematic way of calculating the inverse kinematics of a cluster is shown.

Chapter 3 will introduce the homogenous transformation matrix, as well as give a description of what information it contains. It will then explain how the homogeneous transformation matrix can be obtained for each robot and for each cluster.

Chapter 4 presents a new formula to compute the inverse Jacobian for a cluster space formulation. It then describes an algorithm on how this formula can be implemented.

Chapter 5 describes the Matlab implementation of this new formula. It shows the result of using this new technique on the continuing examples and compares the results to an analytic technique.

Chapter 6 concludes the report with a summary of the findings and possible directions this work can be extended.

Chapter 2: Graph Based Representations of Clusters

In this chapter, the factors that go into cluster variable selection are discussed. A quick review of the homogeneous transform matrix is covered, followed by a proposal of a new way to represent clusters. The cluster is then characterized as graph based representation, and conventions are outlined to do this systematically. Finally, the inverse kinematics for the entire robot cluster are computed by navigating through the graph.

2.1 Determining Suitable Cluster Space Variables

A cluster is a system of “n” robots that are considered a single entity. These robots may be located in the plane, or in three dimensional space. Each robot, n, is free to have “p_n” degrees of spatial and orientation freedom. Since the system is considered a single entity, a single frame of reference, known as a cluster frame, can be assigned to the cluster. While the location of the cluster frame is typically chosen as the average location of all the robots in the cluster, this placement is not a requirement. After selection of a suitable cluster frame, cluster space pose variables are chosen to describe the location and orientation of the cluster frame with respect to the global frame. Other variables are chosen to define the geometry of the cluster, as well as the relative rotation of each robot (usually with respect to the cluster frame).

The act of defining the cluster variables as equations of robot variables is known as finding the Forward Position Kinematics (Eq 1). As mentioned in section 1, this set of expressions is collectively described by the function $KIN(R)$.

$$\vec{C} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_q \end{bmatrix} = KIN({}^G\vec{R}) = \begin{bmatrix} g_1(r_1, r_2, \dots, r_f) \\ g_2(r_1, r_2, \dots, r_f) \\ \vdots \\ g_q(r_1, r_2, \dots, r_f) \end{bmatrix} \quad (\text{Eq 5})$$

The cluster variables chosen do not need to be independent, but collectively, they must span the space. In order to fully span the cluster space of an ‘n’ robot system a minimum of ‘f’ variables are needed to describe the geometry of the system, where f is:

$$f = \sum_{i=1}^n p_n \quad (\text{Eq 6})$$

The total number of cluster space variables is **q**. Having fewer variables than **f**, ($q < f$), leads to the cluster not being fully defined. Having more than **f** variables, ($q > f$), results in one or more of the variables being over-constrained (some variables will be dependent variables).

Since cluster variables are used in describing the geometry, the cluster variables also have implicit constraints. For example, consider the cluster variables L, M, and N, where L is the distance between Robot 1 and Robot 2, M is the distance between Robot 1 and Robot 3, and N is the distance between Robot 2 and Robot 3. As a result of the triangle inequality [24] of geometry, L must be such that $L \leq M+N$. Furthermore, if $L = M+N$, it becomes impossible to change the value of only one variable. See figure 2 below.

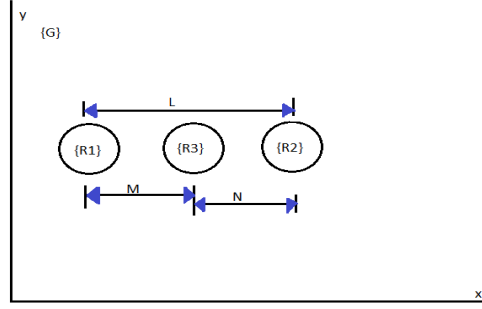


Figure 2: A Cluster of 3 robots, with the distance variables M , N , and L shown

2.2 The Homogeneous Transform in terms of Cluster Space

The homogeneous transform matrix, (also referred to as the Transformation Matrix or the 'T' matrix) has the following form, and describes the position of frame $\{i\}$ relative to frame $\{i-1\}$:

$${}^{i-1}T_i = \begin{bmatrix} {}^{i-1}R_i & {}^{i-1}P_i \\ 0 & 1 \end{bmatrix} \quad (\text{Eq 7})$$

Where ${}^{i-1}R_i$ is a 3x3 Matrix that describes the rotation of frame $\{i\}$ relative to frame $\{i-1\}$ and ${}^{i-1}P_i$ is a 3x1 vector that describes the position of frame $\{i\}$ relative to frame $\{i-1\}$. In terms of cluster space control, these transformation matrices are used in several key ways. First, it can describe the robot's position and orientation relative to the cluster frame which it helps to define. It also describes the relationship between two robots in a leader follower configuration, or two systems in a leader follower configuration (robot following robot, robot following a cluster, cluster following another cluster, etc.) These matrices also describe the relationship between a cluster frame and global frame.

By Euler's Rotation Theorem, any rotation in three dimensional space can be represented as a single rotation about some axis. By this theorem, we can decompose the rotation ${}^{i-1}R_i$ into the product of three rotations:

$${}^{i-1}R_i = {}^{i-1}R_z(\alpha) * {}^{i-1}R_y(\beta) * {}^{i-1}R_x(\gamma) \quad (\text{Eq 8})$$

Which can further be expanded as follows:

$${}^{i-1}R_i = \begin{bmatrix} \cos(\alpha)\cos(\beta) & \cos(\alpha)\sin(\beta)\sin(\gamma) - \sin(\alpha)\cos(\gamma) & \cos(\alpha)\sin(\beta)\cos(\gamma) - \sin(\alpha)\sin(\gamma) \\ \sin(\alpha)\cos(\beta) & \sin(\alpha)\sin(\beta)\sin(\gamma) - \cos(\alpha)\cos(\gamma) & \sin(\alpha)\sin(\beta)\cos(\gamma) - \cos(\alpha)\sin(\gamma) \\ -\sin(\beta) & \cos(\beta)\sin(\gamma) & \cos(\beta)\cos(\gamma) \end{bmatrix} \quad (\text{Eq 9})$$

This is useful to us, as we can now determine the angles for each rotation, creating a vector representing these rotations as follows:

$${}^{i-1}\theta = \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} \quad (\text{Eq 10})$$

Obtaining these angles can be an involved process, based on the hierarchical depth of the rotations.

Since these rotations are orthogonal to each other, rows are linearly independent, and, they can be added and subtracted without affecting each other. This will prove to be very useful in creating computer algorithms that involve robotic systems that have multiple degrees of rotational freedom.

2.3: Graph Based Representations of Clusters

A cluster can be represented in many different ways. Naturally, one way of representing a cluster is to draw a physical configuration of the robot cluster, and then add the cluster variables to the sketch as angle and distance dimensions.

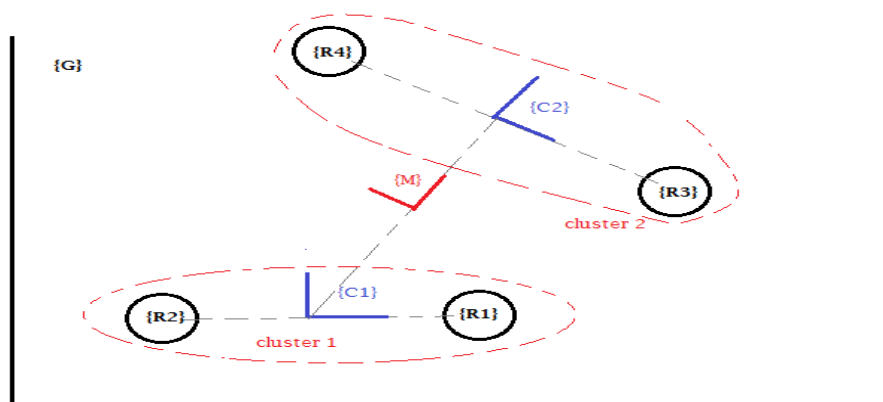


Figure 3: A Physical View of Example 3, a cluster of two 2-robot clusters.

This research proposes a new representation for clusters space formation architecture. This new representation is a formal methodology for defining the cluster based on graph theory. A graph based representation lends itself well to graph theory and to algorithmic information theory, allowing for existing theories and techniques for efficient computation can be applied to the cluster space architecture. Graphs were also chosen as they are capable of capturing hierarchal information needed while using the new propagation based equations.

To represent a cluster as a graph, G , first consider each frame used in the cluster (robot frames, cluster frame, and global frame), as a vertex, V .

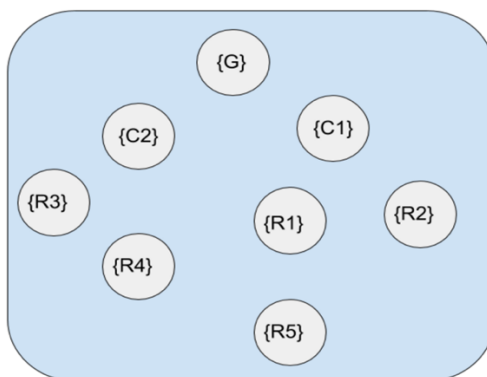


Figure 4: A group of unconnected vertices representing all the frames in a cluster.

Then, for each cluster frame, draw an edge connecting that cluster frame to each robot frame or cluster frame that is used to define that cluster frame's position. This set of edges, E_1 , shall be colored red. A transformation matrix, T , that describes the component frame's position and orientation relative to the cluster must be defined for each of these edges.

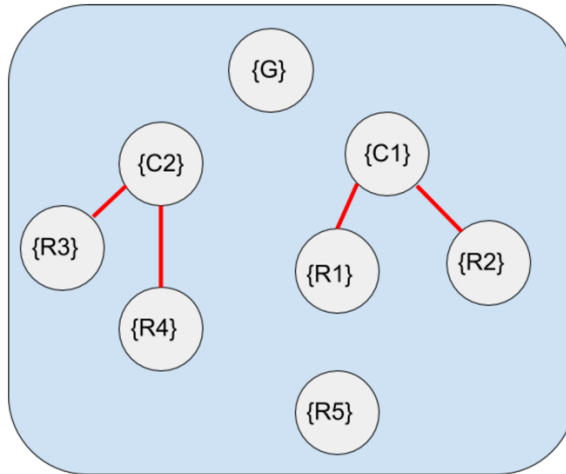


Figure 5: Two 2-Robot Clusters represented by the set of edges E_1 .

Leader follower relationships are then added to the graph, connecting follower vertices to their leader vertices. This set of edges, E_2 , shall be colored blue. A transformation matrix, T , that describes the position and orientation of the frame of the follower relative to its leader must be defined for each of the edges in this set.

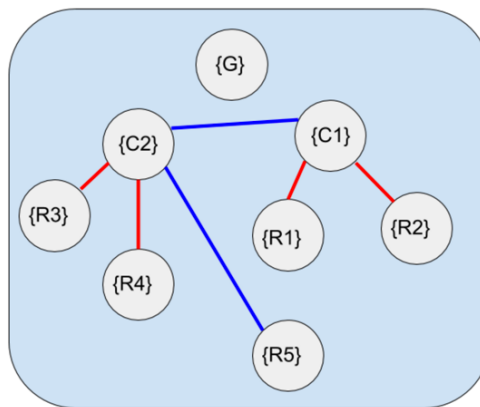


Figure 6: A 2 robot cluster in a leader follower relationship with another 2 robot cluster and a fifth robot in a leader follower relationship with cluster frame C_2

Finally, a single edge, E_3 , is drawn from the primary cluster frame to the global frame. E_3 is to be painted black. A transformation matrix, T , that describes the position and orientation of the primary cluster frame relative to the global frame must be defined for this edge.

This edge also indicates which robot or cluster is leader or follower. The vertex closes to the ground frame is leader, with the other vertex on the other end of the blue edge is the follower.

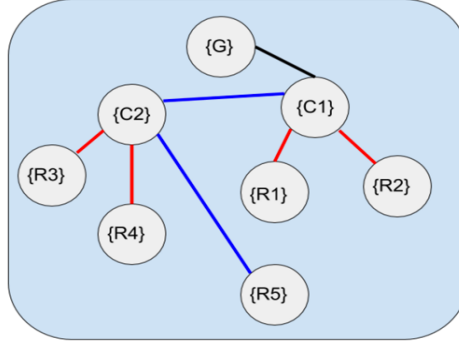


Figure 7: A completed Cluster Graph of a 5 Robot Cluster.

The cluster is then the graph, G , that is described by the set of all vertices and edges described by $G = \{V, E1, E2, E3\}$. By assigning the global frame as the root frame, the graph G is now considered a tree.

By following this convention, we have a standard way of creating and understanding cluster trees. For example, the tree shown in figure 7 can be described as: two systems of 2-robot clusters, with cluster 2 (R3 and R4) following cluster 1 (R1 and R2), and a fifth robot, R5, following cluster 2.

2.4 Determining the Inverse Kinematics from the Graph Based Representations

By defining all necessary T matrices in a cluster, we can compute the inverse kinematics of the system. The inverse position kinematics allow computation of the ‘robot-space’ pose variables, the elements of \vec{R} , as a function of the cluster space variables, the elements of \vec{C} . Collectively, this set of equations is known as the Inverse Kinematic Equations. This can be defined as the function $INVKIN(\vec{C})$:

$$\vec{R} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_f \end{bmatrix} = INVKIN({}^G\vec{C}) = \begin{bmatrix} h_1(c_1, c_2, \dots, c_q) \\ h_2(c_1, c_2, \dots, c_q) \\ \vdots \\ h_3(c_1, c_2, \dots, c_q) \end{bmatrix} \quad (\text{Eq 11})$$

In order to compute the inverse Kinematics, not only must we compute the individual transformation matrices from each vertex to its parent vertex, but the entire transformation from each robot node frame to the global frame must be determined. This can be done by performing a depth first search from the root node down to the robot nodes. Once a robot node is found, the ‘ T ’ matrices defined along the path from the root node to the robot node are then multiplied together.

This, in fact, implements a technique proposed by Dr. Chris Kitts to compute inverse kinematics. In this proposal, the inverse kinematics can be found by finding the product of intermediate homogeneous transforms, such that the homogeneous transform for any robot, i , is found by:

$${}^G_iT = {}^G_1T * (\prod_{j=2}^d {}^{C_{j-1}}_{C_j}T) * {}^{C_d}_iT \quad (\text{Eq 12})$$

Where d is the number of hierarchical frame steps between $\{G\}$ and $\{i\}$, and C_n is the n^{th} local cluster frame between $\{G\}$ and $\{i\}$. In the case that $n < 2$, the product term vanishes. In the case of a 1 robot system, it is assumed that the robot is following some cluster frame, even if that frame is incident with the robot frame, resulting in a T matrix which is equal to the identity matrix.

The result of equation 12 will also have the form of a homogeneous transform matrix. The resulting matrix will yield:

$${}^G_iT = \begin{bmatrix} {}^G_iR & {}^G_iP \\ 0 & 1 \end{bmatrix} \quad (\text{Eq 13})$$

Where G_iP is the position vector of robot i relative to the global frame, and G_iR is the relative rotation of robot i with respect to the global frame.

From this matrix, the position \hat{x} , \hat{y} , and \hat{z} for each robot i can be recovered from G_iP . Furthermore, G_iR can be decomposed into $R_z(\alpha)$, $R_y(\beta)$, and $R_x(\gamma)$ for each robot i . Finding these yields the inverse kinematic equations of the system.

This shows that the tree structure described in section 2.3 accurately produces the inverse kinematics for a given cluster. This result lends credibility that this particular method for describing a cluster as a graph is valid.

2.5A few notes on the Tree Representation of a cluster space formulation

Just like with serial manipulator, this method is much simpler than algebraically rearranging the forward kinematic equations to isolate the individual robot variables. However, this method allows for the possibility of some information contained in the forward kinematic equations to be lost. In other words, if one were to try to isolate the robot variables from the system of equations obtained from the propagation technique for the inverse kinematics, then multiple solutions can be obtained for some of the robot variables. This can possibly be avoided with a more rigorous method of selecting cluster variable (see future work in Section 6.2).

There are other similarities of this technique to the propagation technique used by serial manipulators. If a homogeneous transformation matrix, A_BT , is an edge that describes a path from vertex $\{A\}$ to vertex $\{B\}$, then the matrix B_AT , the edge $E2$ that is a path from $\{B\}$ to $\{A\}$, can be found by inverting A_BT . This frame transformation technique is also used in serial and parallel manipulators.

$${}^B_AT = {}^A_BT^{-1} \quad (\text{Eq 14})$$

Using equation 12 and 14, it is possible, although unnecessary, to create an edge from any vertex in the graph to each vertex in the graph, including itself. This may be useful if you need to describe one robot in a cluster relative to another.

Furthermore, it is possible for a robot to have two leaders. For example, in figure 7, R3 can be following R1 at a specific distance and angle, but follow the orientation of R2. Another example is that robot R3 can be told to stay a distance $d1$ from R2, and a distance $d2$ from R3. If this occurs, the 'T' matrices for each blue edge will be ill-defined. Configurations like those shown in figure 7 do not present a problem mathematically, and can be solved with incomplete HTM's like equations 15 and 16. The trick is to propagate from R3 to R1 to C1 to G, and then from R3 to R2 to C1 to G. Then, toss out any results containing xx and yy . The rest of this research will assume that each follower has only one leader.

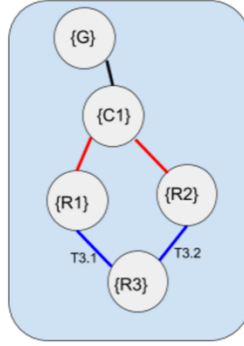


Figure 8: A Robot with two leaders.

$$T_{3.1} = \begin{bmatrix} \cos(xx) & -\sin(xx) & l * \cos(\hat{\theta}_c) \\ \sin(xx) & \cos(xx) & l * \sin(\hat{\theta}_c) \\ 0 & 0 & 1 \end{bmatrix} \quad (\text{Eq 15})$$

$$T_{3.2} = \begin{bmatrix} \cos(A) & -\sin(A) & yy \\ \sin(A) & \cos(A) & yy \\ 0 & 0 & 1 \end{bmatrix} \quad (\text{Eq 16})$$

Chapter 3: Jacobian Propagation Algorithm

This section presents the given formulas used for determining the inverse Jacobian of a cluster. This also lays the mathematical foundation for the implementation of these formulae. A particular way to implement these formulas is then presented. This chapter concludes with a run time analysis of this particular implementation.

3.1 The Inverse Jacobian Algorithm

The Robot Space position variables, \vec{R} , and the Cluster Space variables, \vec{C} , are mapped by the functions *KIN* (Eq 1) and *INVKIN* (Eq 2). To map the velocities from Robot Space, \vec{R} , to Cluster Space, \vec{C} , the matrix of partial derivatives, $J(R)$, known as the Jacobian Matrix is used. Similarly, the inverse Jacobian, $J^{-1}(R)$, transforms Cluster Space velocities to robot space velocities.

$$J^{-1}(C)\vec{C} = \begin{bmatrix} \frac{dh_1}{dc_1} & \dots & \frac{dh_1}{dc_q} \\ \vdots & & \vdots \\ \frac{dh_f}{dc_1} & \dots & \frac{dh_f}{dc_q} \end{bmatrix} \begin{bmatrix} \dot{c}_1 \\ \dot{c}_2 \\ \vdots \\ \dot{c}_q \end{bmatrix} \quad (\text{Eq 17})$$

For cluster space control, the inverse Jacobian historically has been computed by taking partial derivatives of the inverse kinematics. This process is a time consuming and computationally expensive process. There is a need for faster calculation of the Jacobian for uses in high speed systems.

It was proposed by Dr. Kitts that each row of the matrix, ∇h_n , is a robot space velocity vector that can be developed by performing a velocity propagation analysis for each robot n . The propagation starts with determining the velocity of a robot, n , with a fixed frame $\{n\}$ relative to its local cluster frame $\{i\}$. Both the linear velocity of robot n relative to frame $\{i\}$, ${}^i_n\vec{V}$, and its angular velocity, ${}^i_n\vec{\omega}$, must be determined and written using cluster space variables. These velocities are then propagated from frame to frame (from cluster to leader cluster or to parent cluster). The propagation continues until the last frame; that of global frame $\{G\}$. The result of performing these propagations for each robot leads to a system of linear equations in cluster variable that are assembled to form the inverse Jacobian. This results in mapping cluster space velocities to robot space velocities.

$$J^{-1}(C)\vec{C} = \begin{bmatrix} \nabla h_1 \\ \nabla h_2 \\ \vdots \\ \nabla h_f \end{bmatrix} = \begin{bmatrix} {}^G_1\vec{V} \\ {}^G_1\vec{\omega} \\ \vdots \\ {}^G_n\vec{V} \\ {}^G_n\vec{\omega} \end{bmatrix} \quad (\text{Eq 18})$$

To find the linear and angular velocities of robot n with a fixed object frame $\{n\}$ in frame $\{i\}$, the rate of change of their linear and angular position can be taken as follows:

$${}^i_n\vec{V} = \dot{{}^i_n\vec{P}} + ({}^i_n\vec{\omega} \times {}^i_n\vec{P}) \quad (\text{Eq 19})$$

$${}^i_n\vec{\omega} = ({}^i_n\dot{\theta}) \quad (\text{Eq 20})$$

Where:

- ${}^i_n\vec{V}$ = The total linear velocity of the frame, n, relative to frame i
- ${}^i_n\vec{P}$ = The position of the frame, n, relative to frame i
- $\dot{{}^i_n\vec{P}}$ = The derivative of ${}^i_n\vec{P}$ or $\nabla_n {}^i_n\vec{P}$
- ${}^i_n\theta$ = The angle of the two argument arctangent of the position vector, ${}^i_n\vec{P}$
- ${}^i_n\vec{\omega}$ = the derivative of ${}^i_n\theta$, or $\nabla_n {}^i_n\theta$, or ${}^i_n\dot{\theta}$

Since the linear and angular velocities of the frame {n} in frame {i} are known, then it is possible to calculate the velocities of that frame relative to frame {i-1} as follows:

$${}^{i-1}_n\vec{V} = {}^{i-1}_i\vec{V} + {}^{i-1}_iR (\dot{{}^i_n\vec{P}} + ({}^{i-1}_n\vec{\omega} \times {}^i_n\vec{P})) \quad (\text{Eq 21})$$

$${}^{i-1}_n\vec{\omega} = {}^{i-1}_i\vec{\omega} + {}^{i-1}_iR * {}^i_n\vec{\omega} \quad (\text{Eq 22})$$

Where:

- ${}^{i-1}_n\vec{V}$ = The total linear velocity of the object n relative to frame {i-1}
- ${}^{i-1}_i\vec{V}$ = The total linear velocity of frame {i} relative to frame {i-1}
- ${}^{i-1}_iR$ = The rotation matrix that describes the fixed rotation of frame {i} relative to frame {i-1}
- ${}^{i-1}_i\vec{\omega}$ = the angular velocity of frame {i} about frame {i-1}
- ${}^{i-1}_n\vec{\omega}$ = the angular velocity of the object n relative to frame {i-1}

Therefore, for any robot, n, with hierarchical depth m, these equations can be applied recursively from the robot through the hierarchal path of frames back to the global frame {G}. Doing so results in the equations:

$${}^G_n\vec{V} = \sum_{k=1}^m (\prod_{j=1}^k {}^{j-1}_jR) ({}^{k-1}_k\dot{\vec{P}} + ({}^G_k\vec{\omega} \times {}^{k-1}_k\vec{P})) \quad (\text{Eq 23})$$

$${}^G_k\vec{\omega} = \sum_{p=1}^k (\prod_{i=1}^p {}^{i-1}_iR) ({}^{i-1}_i\vec{\omega}) \quad (\text{Eq 24})$$

Where:

- ${}^G_n\vec{V}$ = The total linear velocity of robot n relative to global frame
- ${}^G_k\vec{\omega}$ = The total angular velocity of frame k relative to the global frame
- ${}^{j-1}_jR$ = The rotation matrix that describes the fixed rotation of frame {j} relative to {j-1}
- ${}^{k-1}_k\vec{P}$ = The instantaneous position of frame {k} relative to frame {k-1}
- ${}^{k-1}_k\dot{\vec{P}}$ = The rate of change of instantaneous position of frame {k} relative to frame {k-1}

For equations 23 and 24, j is the number of steps that the current frame is away from the cluster frame, and m depth level of robot n. It is worth noting that when j = 1, ${}^{j-1}_jR$ becomes the identity matrix, and when k=1, ${}^G_k\vec{\omega}$, becomes a zero vector.

By applying these formulas for each robot in a given cluster, we can determine the inverse Jacobian of that entire system. A derivation of these formulas can be found in Appendix C.

3.2 An implementation of the Formula

The recursive nature of the formulae in section 3.1 lend themselves to creating an algorithm for execution. In an attempt to do this as quickly and as efficiently as possible, the Cluster Tree Inverse Jacobian algorithm was created. This algorithm is outlined as follows:

```
invJ = invjacfxn(clusterTree)
```

Initialization

```
{  
  Index all nodes on clusterTree  
  Create list of cluster variables  
  Create list of all robot nodes  
  Create empty invJ matrix to populate  
}
```

Main Loop

```
{  
  For each node listed in list of all robot nodes  
    Linear Velocity Propagation Subroutine  
    Save Result to invJ matrix  
    Angular Velocity Propagation Subroutine  
    Save Result to invJ matrix  
  End  
}
```

Termination

```
{  
  Terminate after all elements in list of all robot nodes has been visited  
  Display final invJ Matrix showing the inverse Jacobian  
}
```

Both subroutines in the Main Loop are rather involved sequences which require traversal of the Cluster Tree that is taken as an input. A detailed explanation of the initialization, main loop, and termination steps are as follows:

Initialization

- 1) Let us start by creating a cluster tree as listed in section 2.3. This tree must have all T matrices defined and recorded as edges. Let all nodes in the tree begin colored white, with all edges retaining their previous coloration. Nodes on the tree are indexed by a breadth first search algorithm.
- 2) Create a list containing the derivatives of each of the cluster variables. (*'cluster_var'*)
- 3) Create a list of all the robot nodes that are present in the tree, as well as their index in the tree. (*'robot_node'*)

- 4) We shall also create a blank matrix $m \times n$ in size named J^{-1} . The size of m is the sum of all the degrees of freedom of all the robots in a cluster, and n is determined by the number of cluster variables. Initialize all entries in this matrix to 0.

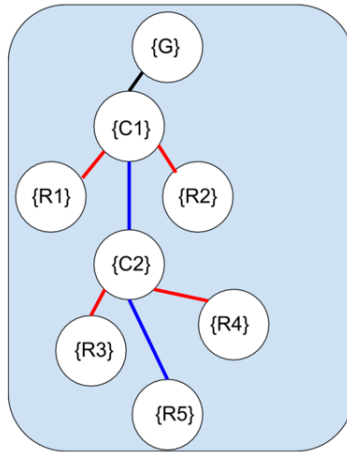


Figure 9: Initial tree with all nodes colored white.

Main Loop

- 1) Visit the first element in the 'robot_node' list, and color the node yellow.

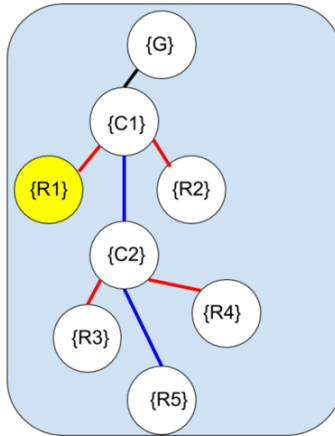


Figure 10: Tree with current node colored yellow.

- 2) Perform Linear Velocity Propagation Subroutine

2.1) Create a variable, $'varA' = \begin{bmatrix} expr1 \\ expr2 \\ expr3 \end{bmatrix}$. Each expression in this variable is automatically factored by the list 'cluster_var'. Initialize all entries to 0.

- 2.2) Compute the conjugate rotation of the yellow node from the global frame.
 2.2.1) Extract the local rotation of current frame from the parent frame by extracting it from the HTM. Save this as the variable 'RR'.

2.2.2) Update the current node to the parent node, coloring the new node yellow, and the old node red. Repeat step 2.2.1, adding the previously extracted rotation to the new rotation ($RR' = RR' + \text{'new result'}$).

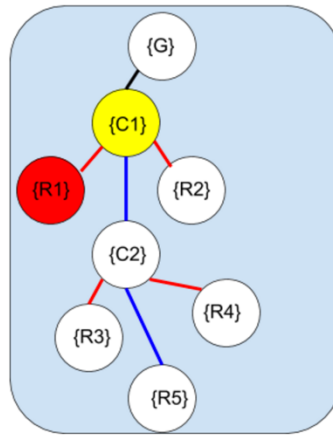


Figure 11: Cluster tree with explored node colored red and current node yellow

2.2.3) Repeat until root node is current node. Save result as RR' . Turn this node red.

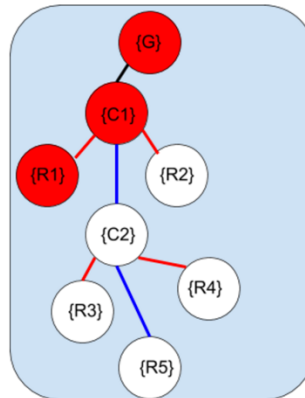


Figure 12: Cluster tree with all nodes from leaf to root explored once.

2.3) Return to red node with the highest index, and make this node the current node, by changing the color of the node to yellow. Extract the local position of this node from the HTM, and get the local velocity by partial differentiation.

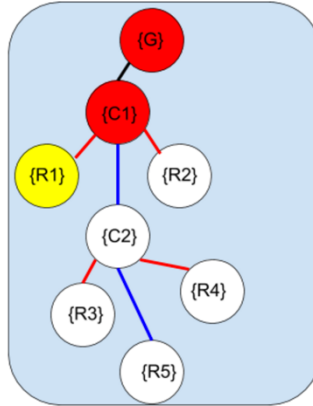


Figure 13: Returning to the robot node of the current subroutine.

2.4) Rotate the local velocity in step 2.3 and rotate it by the 'RR' calculated in step 2.2

2.5) Find the global angular velocity by differentiating the variable 'RR' from step 2.2. Then take the cross product of that with the position of the current node relative to its parent node.

2.6) Rotate the cross product calculated in step 2.5 by 'RR' determined in step 2.2

2.7) Update the value of 'varA' to include the the results of steps 2.4 and 2.6 as follows:
 $\text{'varA'} += \text{step2.4} + \text{step2.6}$

2.8) Update the current node on the tree by changing the color of the current node to the color green and its parent to yellow, thus making the parent the new current node.

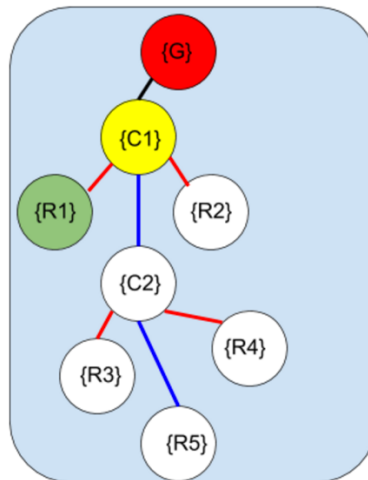


Figure 14: Cluster Tree with Explored nodes are green and current node is yellow.

2.9) Repeat steps 2.2 thru 2.8 until the root node is reached.

2.10) Get the local position of the root node and differentiate it.

2.11) Add the result to 'varA'. ('varA' += 'step2.10'). Color the root node green.

- 3) Save the result of step 2.11 to the first rows of the matrix named J^{-1} below where varb was last stored. If this is the first iteration of the loop, save to the top rows of J^{-1} .
- 4) Perform Angular Velocity Propagation Subroutine

- 4.1) Create a variable, $var_b = \begin{bmatrix} expr1 \\ expr2 \\ expr3 \end{bmatrix}$. Each expression in this variable is automatically split and sorted by the list 'cluster_var'. Initialize all entries to 0.
- 4.2) Perform a search to find the green node with the largest index. Make this node the current node (colour it yellow).

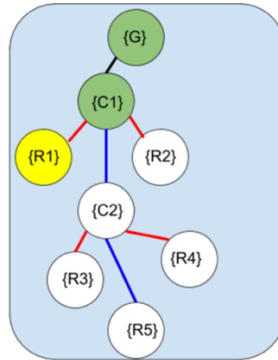


Figure 15: Returning to robot node being computed and mark as current node

- 4.3) Calculate 'RR' for the current node similarly to step 2.2. Use red again.
- 4.4) Calculate global angular velocity by differentiating the result of step 3.3. Save the result to 'varB'.

- 5) Save the 'varB' to rows of the matrix named J^{-1} directly below where varA was stored.
- 6) Return to red node with the highest index and color this node black. Restore all non-black nodes to the color white.

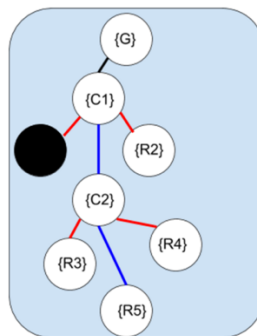


Figure 16: A cluster tree with a single fully explored robot node

- 7) Go to the following element listed in the 'robot_node' list and color it yellow.

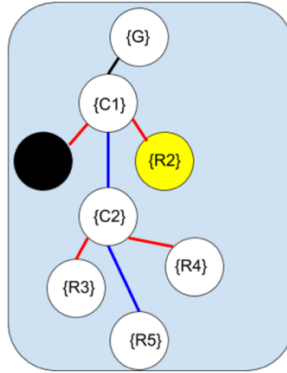


Figure 17: A fully explored robot node and a new robot node selected.

- 8) Repeat steps 2 through 7 of Main Loop

Termination

- 1) Continue Repeating steps 2 through 7 until all nodes for elements in the list 'robot_node' are colored black.

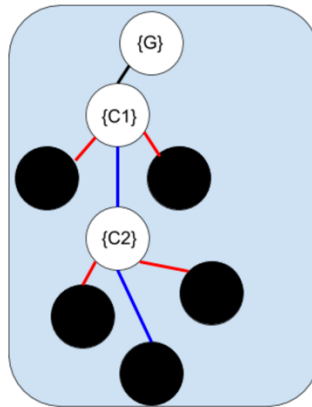


Figure 18: A fully explored cluster tree.

Upon completion of this algorithm, the inverse Jacobian will be obtained and saved into the matrix J^{-1} .

This implementation only supports non-cyclic graphs, and clusters that are defined in a Cartesian coordinate system

3.3 Run Time Analysis of this algorithm

Scalability of this algorithm is important to consider with respect to the number of cluster variables, number of robots, and hierarchical depths present in a cluster. To analyze how this algorithm would respond to an increase in any of these conditions, runtime analyses were performed, and the growth rates expressed in Big O notation.

It was determined that the runtime increases linearly to an increase in number of cluster variables. In the algorithm above, there are no loops that are affected by the length of the list 'cluster_var'. However, the length of 'cluster_var' affects how 'varA' and 'varB' are factored, and how many partial derivatives need to be calculated for each step involving a differentiation. If we assume a parsing function scales linearly with the number of delimiters, and if we assume that taking partial derivatives scales linearly with the number of partials taken, then the above algorithm has a runtime of $O(n)$ with respect to number of cluster variables.

The algorithm also scales linearly with an increase in the number of robots. The number of robots in the cluster directly affects the length of the list 'robot_node'. For each element in this list, subroutine A is performed exactly once, and so is subroutine B. Therefore, the algorithm above scales linearly with the number of robots. Therefore, this algorithm scales like $O(n)$ with respect to number of robots in the cluster.

An exponential increase occurs with an increase in hierarchal depth. For any vertex at depth d , step 2.2 of the linear velocity propagation subroutine, as well as all of the angular velocity propagation subroutine, needs to compute with the $d-1$ vertices above it in the tree already solved. Since the algorithm didn't save the previous result, it recomputed this information every time the current node updated to the parent node. Therefore, this algorithm scales like $O(n^2)$ with respect to overall hierarchal depth of the cluster.

Chapter 4: Matlab Implementation of Algorithm

This section will describe a Matlab toolbox that was created to test the algorithm stated in Chapter 3. It will outline each file used in each of the folders of the toolbox. The first major folder, Cluster_Builder, contains files that guide the user in creating a cluster tree to use for their custom cluster space configuration of robots. The second major folder contains files that compute the inverse kinematics and the inverse Jacobian analytically by taking partial derivatives of the inverse kinematics. The third folder contains a list of examples used for verifying the algorithm described in section three. Finally, the fourth major folder contains files that implement the algorithm stated above. The Archive folders contained deprecated files, and the @tree folder contains the definition of the tree class.

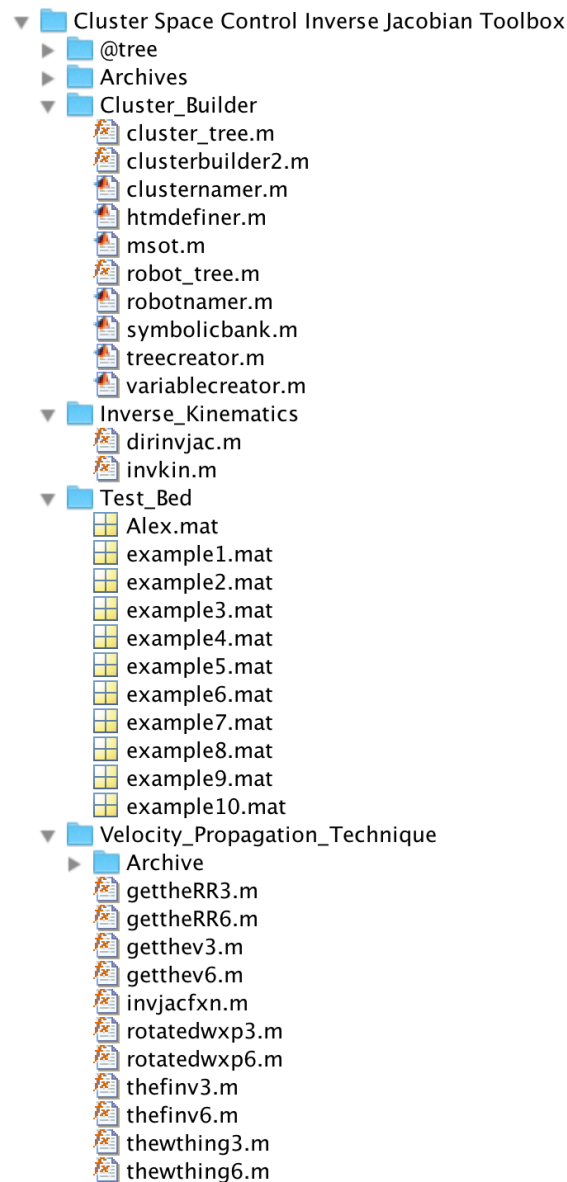


Figure 19: Directory of the Cluster Space Control Inverse Jacobian Toolbox

4.1 The Cluster_Builder Folder

The files in this folder allows the user of the toolbox to enter information about their custom clusters, and then compute the inverse Jacobian for that cluster using either the direct approach, or the propagation based approach.

To build a cluster tree using the files in this folder, one must manually create a cluster tree and have all edges defined and ready to be inputted. The user must also have downloaded the tree class from the MathWorks website, as it is not a standard class. This class was created by Jean-Yves Tinevez on 13 March 2012 and updated on 18 Nov 2015 It can be downloaded from the link: <http://www.mathworks.com/matlabcentral/fileexchange/35623-tree-data-structure-as-a-matlab-class>

The file clusterbuilder2.m is then given the number of robot nodes and number of cluster nodes as input, and then guides the user on how to enter all the cluster information into a single Matlab variable. Table 4 lists all the files in this folder, and provides a description of what each file does. Figure 19 and 20 shows how this function looks on screen.

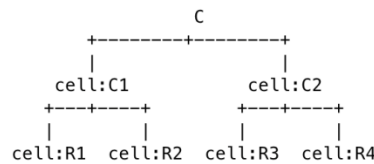
Table 1: A list of all the files in the Cluster_Builder folder

File Name	Description
<i>Clusterbuilder2.m</i>	This is the main function that calls all the other subroutines. It takes the number of robot nodes and the number of cluster nodes as input, and the final result in a variable containing the cluster information. The function command is [obj_out] = <i>clusterbuilder2</i> (num_robots,num_clusters)
<i>Robotnamer.m</i>	This program names all the robots as R1, R2, R3. Etc... The name of the robot at node (n) can later be renamed by using the command example.txt_tree=example.txt_tree.set(n,'newrobotname').
<i>Clusternamer.m</i>	This program names all the clusters as C, C1, C2, C3.... The name of the cluster at node 'm' can later be renamed by using the command example.txt_tree=example.txt_tree.set('m','newclustername').
<i>Variablecreator.m</i>	Asks the user to list all variable names that will be used to describe geometric features of the cluster. It stores this information in a vector.
<i>Symbolicbank.m</i>	Loads a bank of symbolic variables to memory. It also loads all assumptions on symbolic variables.
<i>Treecreator.m</i>	Asks user to arrange the robot nodes and cluster nodes into an arborescence. (Tree structure, not a graph) that spans all nodes with only one root. Calls subroutines <i>robot_tree.m</i> and <i>cluster_tree.m</i>
<i>Robot_tree.m</i>	Subroutine of <i>treecreator.m</i> . It adds robot nodes to the tree
<i>Cluster_tree.m</i>	Subroutine of <i>treecreator.m</i> It adds cluster nodes to the tree.
<i>Htmdefiner.m</i>	asks the user to enter the homogeneous transformation matrix that describes the position and orientation of each node relative to its parent.
<i>Msot.m</i>	Compiles all the information into a single structure containing all the attributes listed in table 4.1

```

>> [example3] = clusterbuilder2(4,2)
What Cluster variables will you use?
Please state your variables with no commas
Example: a b c d xc yc phi1
>> xc yc tc l m n tc1 tc2 phi1 phi2 phi3 phi4
Let us now start creating our cluster tree
How many robot children does C have?
: 0
How many cluster children does C have
: 2
How many robot children does C1 have?
: 2
How many cluster children does C1 have
: 0
How many robot children does C2 have?
: 2
How many cluster children does C2 have
: 0

```



The Cluster tree we created is visible above.

Figure 20: User entering tree information for example3 into *clusterbuilder2.m*

```

Enter the 3x3 or 4x4 T matrix from cluster frame to Ground Frame
>> [cos(tc) -sin(tc) xc; sin(tc) cos(tc) yc; 0 0 1]
What is the Transformation from C1 to C
>> : [cos(tc1) -sin(tc1) l; sin(tc1) cos(tc1) 0; 0 0 1]
What is the Transformation from C2 to C
>> : [cos(tc2) -sin(tc2) -l; sin(tc2) cos(tc2) 0; 0 0 1]
What is the Transformation from R1 to C1
>> : [cos(phi1) -sin(phi1) m; sin(phi1) cos(phi1) 0; 0 0 1]
What is the Transformation from R2 to C1
>> : [cos(phi2) -sin(phi2) -m; sin(phi2) cos(phi2) 0; 0 0 1]
What is the Transformation from R3 to C2
>> : [cos(phi3) -sin(phi3) n; sin(phi3) cos(phi3) 0; 0 0 1]
What is the Transformation from R4 to C2
>> : [cos(phi4) -sin(phi4) -n; sin(phi4) cos(phi4) 0; 0 0 1]

example3 =

  robotnodes: [4 5 6 7]
  clusternodes: [1 2 3]
  htm_tree: [1x1 tree]
  txt_tree: [1x1 tree]
  var: [1x12 sym]

```

Figure 21: User entering HTM matrices for each edge on the tree of example3

Once the user finishes the entry sequence for that cluster, the *clusterbuilder2.m* function outputs a variable as seen in figure 22 with the attributes listed in table 2. The attributes *example.propinvjac*, *example.inv_kin*, and *example.dirinvjac* are not initially saved in this variable. They are only created after the inverse kinematics or the respective Jacobian function is used on the variable.

4.2 The Inverse Kinematics Folder

The second major folder of the Toolbox contains files that execute the algorithm described in section 3. This folder contains files that work for clusters with 6 degrees of freedom, and an optimized set of subroutines for working with clusters that are limited to the plane. Table 3 is a list of all the files and which steps in the algorithm that they refer to. Figures 23 and 24 show the result of using these functions.

Table 3: Description of all Files in the Inverse Kinematics folder

File Name	Description
<i>invkin.m</i>	This file computes the inverse kinematics for each robot by tracing a path from each robot node back to the root node. It then multiplies the homogeneous transform matrices from root back down to the leaf nodes (robot nodes). It then saves the results as a variable attribute, example.inv_kin
<i>dirinvjac.m</i>	This file first calls the program <i>invkin.m</i> to compute the inverse kinematics of each robot. Each element in the vector example.inv_kin is It then differentiated by each symbolic element in the vector example.syms. The resulting array is saved as the variable attribute example.dirinvjac

```
>> [inv_kin,example3]=invkin(example3)

inv_kin =

xc + m*cos(tc + tc1) + l*cos(tc)
yc + m*sin(tc + tc1) + l*sin(tc)
      phi1 + tc + tc1
xc - m*cos(tc + tc1) + l*cos(tc)
yc - m*sin(tc + tc1) + l*sin(tc)
      phi2 + tc + tc1
xc + n*cos(tc + tc2) - l*cos(tc)
yc + n*sin(tc + tc2) - l*sin(tc)
      phi3 + tc + tc2
xc - n*cos(tc + tc2) - l*cos(tc)
yc - n*sin(tc + tc2) - l*sin(tc)
      phi4 + tc + tc2
```

Figure 23: The function *invkin.m* being used on *example3* variable

```

>> [ dirinvjac, example3 ] = dirinvjac(example3)

dirinvjac =

[ 1, 0, -m*sin(tc + tc1) - l*sin(tc), cos(tc), cos(tc + tc1), 0, -m*sin(tc + tc1), 0, 0, 0, 0, 0]
[ 0, 1, m*cos(tc + tc1) + l*cos(tc), sin(tc), sin(tc + tc1), 0, m*cos(tc + tc1), 0, 0, 0, 0, 0]
[ 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0]
[ 1, 0, m*sin(tc + tc1) - l*sin(tc), cos(tc), -cos(tc + tc1), 0, m*sin(tc + tc1), 0, 0, 0, 0, 0]
[ 0, 1, l*cos(tc) - m*cos(tc + tc1), sin(tc), -sin(tc + tc1), 0, -m*cos(tc + tc1), 0, 0, 0, 0, 0]
[ 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0]
[ 1, 0, l*sin(tc) - n*sin(tc + tc2), -cos(tc), 0, cos(tc + tc2), 0, -n*sin(tc + tc2), 0, 0, 0, 0]
[ 0, 1, n*cos(tc + tc2) - l*cos(tc), -sin(tc), 0, sin(tc + tc2), 0, n*cos(tc + tc2), 0, 0, 0, 0]
[ 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0]
[ 1, 0, n*sin(tc + tc2) + l*sin(tc), -cos(tc), 0, -cos(tc + tc2), 0, n*sin(tc + tc2), 0, 0, 0, 0]
[ 0, 1, -n*cos(tc + tc2) - l*cos(tc), -sin(tc), 0, -sin(tc + tc2), 0, -n*cos(tc + tc2), 0, 0, 0, 0]
[ 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0]

example3 =

robotnodes: [4 5 6 7]
clusternodes: [1 2 3]
htm_tree: [1x1 tree]
txt_tree: [1x1 tree]
var: [1x12 sym]
inv_kin: [12x1 sym]
dirinvjac: [12x12 sym]
propinvjac: [12x12 sym]

```

Figure 24: The function `dirinvjac.m` being used on `example3` variable

4.3 The Test_Bed Folder

The third major folder of the Toolbox is full of example clusters. Each sample cluster is saved as a variable, and each attribute of the variable contains information that describes the cluster. A full list of all the attributes can be found in table 2.

Each of the examples saved in the test bed represent a different type of cluster. These examples vary in number of robots, number of degrees of freedom, hierarchical depth of robots, and clusters of different types of configurations. To load an example to the active workspace, use the command `load('example.mat')`. All examples included in this folder are listed in table 4.

Figures 25 and 26 show the physical view and graphical view of Example3. The Cluster frame, C, is determined by two subclusters, C1 and C2. Robot1 and Robot2 form the subcluster C1, and Robot3 and Robot4 form subcluster C2. Equations 25 and 33 give the forward and inverse kinematics of the system respectively. Equations 26-32 give the HTM's for each edge needed for the graph.

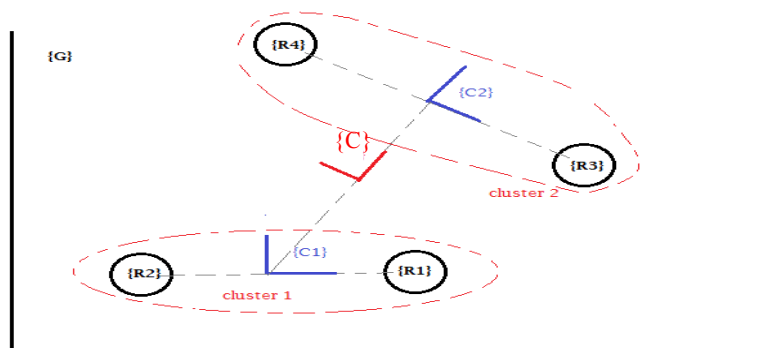


Figure 25: A Physical View of Example3, a cluster of two 2-robot clusters.

$$\begin{bmatrix} \hat{x}_c \\ \hat{y}_m \\ \hat{\phi}_1 \\ \hat{\phi}_2 \\ \hat{\phi}_3 \\ \hat{\phi}_4 \\ \hat{\theta}_c \\ \hat{l} \\ \hat{m} \\ \hat{n} \\ \hat{\theta}_{c1} \\ \hat{\theta}_{c2} \end{bmatrix} = \begin{bmatrix} \frac{(x_1 + x_2 + x_3 + x_4)}{4} \\ \frac{(y_1 + y_2 + y_3 + y_4)}{4} \\ \hat{\theta}_1 - \hat{\theta}_{c1} - \theta_m \\ \hat{\theta}_2 - \hat{\theta}_{c1} - \hat{\theta}_m \\ \hat{\theta}_3 - \theta_{c2} - \theta_m \\ \hat{\theta}_4 - \hat{\theta}_{c2} - \hat{\theta}_m \\ \text{atan2}\left((y_m - y_{c1}), \left(x_m - \frac{(x_1 + x_2)}{2}\right)\right) \\ \sqrt{\left(\frac{(y_1 + y_2)}{2} - y_m\right)^2 + \left(\frac{(x_1 + x_2)}{2} - x_m\right)^2} \\ \sqrt{\left(\frac{(y_1 + y_2)}{2} - y_1\right)^2 + \left(\frac{(x_1 + x_2)}{2} - x_1\right)^2} \\ \sqrt{\left(\frac{(y_3 + y_4)}{2} - y_1\right)^2 + \left(\frac{(x_3 + x_4)}{2} - x_1\right)^2} \\ \text{acos}\left(\frac{((y_m - y_1)^2 + (x_m - x_1)^2 - 4\hat{l}^2 - \hat{m}^2)}{4lm}\right) \\ \text{acos}\left(\frac{((y_m - y_3)^2 + (x_m - x_3)^2 - 4\hat{l}^2 - \hat{n}^2)}{4ln}\right) \end{bmatrix} \quad (\text{Eq 25})$$

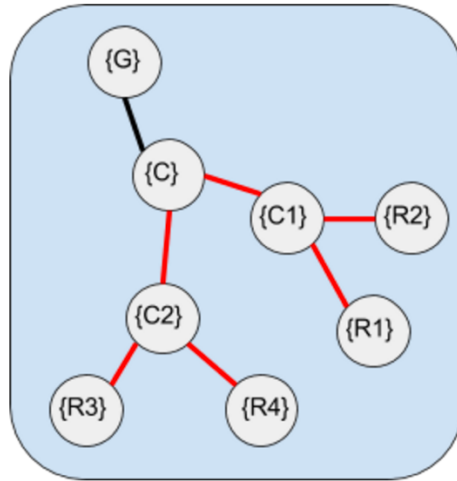


Figure 26: A Graph representing a cluster of clusters. Here cluster {C1} and {C2} form a single cluster {C}.

$${}^G_c T = \begin{bmatrix} \cos(\hat{\theta}_m) & -\sin(\hat{\theta}_m) & x_c \\ \sin(\hat{\theta}_m) & \cos(\hat{\theta}_m) & y_c \\ 0 & 0 & 1 \end{bmatrix} \quad (\text{Eq 26})$$

$${}^c_{c1} T = \begin{bmatrix} \cos(\hat{\theta}_{c1}) & -\sin(\hat{\theta}_{c1}) & l \\ \sin(\hat{\theta}_{c1}) & \cos(\hat{\theta}_{c1}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (\text{Eq 27})$$

$${}^c_{c2} T = \begin{bmatrix} \cos(\hat{\theta}_{c2}) & -\sin(\hat{\theta}_{c2}) & -l \\ \sin(\hat{\theta}_{c2}) & \cos(\hat{\theta}_{c2}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (\text{Eq 28})$$

$${}^{c1}_{R1} T = \begin{bmatrix} \cos(\hat{\phi}_1) & -\sin(\hat{\phi}_1) & m \\ \sin(\hat{\phi}_1) & \cos(\hat{\phi}_1) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (\text{Eq 29})$$

$${}^{c1}_{R2} T = \begin{bmatrix} \cos(\hat{\phi}_2) & -\sin(\hat{\phi}_2) & -m \\ \sin(\hat{\phi}_2) & \cos(\hat{\phi}_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (\text{Eq 30})$$

$${}^{c2}_{R3} T = \begin{bmatrix} \cos(\hat{\phi}_3) & -\sin(\hat{\phi}_3) & n \\ \sin(\hat{\phi}_3) & \cos(\hat{\phi}_3) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (\text{Eq 31})$$

$${}^{c2}_{R4} T = \begin{bmatrix} \cos(\hat{\phi}_4) & -\sin(\hat{\phi}_4) & -n \\ \sin(\hat{\phi}_4) & \cos(\hat{\phi}_4) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (\text{Eq 32})$$

$$\begin{bmatrix} x_1 \\ y_1 \\ \hat{\theta}_1 \\ x_2 \\ y_2 \\ \hat{\theta}_2 \\ x_3 \\ y_3 \\ \hat{\theta}_3 \\ x_4 \\ y_4 \\ \hat{\theta}_4 \end{bmatrix} = \begin{bmatrix} m * \cos(\hat{\theta}_{c1} + \hat{\theta}_m) + l * \cos(\hat{\theta}_m) + x_c \\ m * \sin(\hat{\theta}_{c1} + \hat{\theta}_m) + l * \sin(\hat{\theta}_m) + y_c \\ \hat{\theta}_m + \hat{\theta}_{c1} + \hat{\phi}_1 \\ -m * \cos(\hat{\theta}_{c1} + \hat{\theta}_m) + l * \cos(\hat{\theta}_m) + x_c \\ -m * \sin(\hat{\theta}_{c1} + \hat{\theta}_m) + l * \sin(\hat{\theta}_m) + y_c \\ \hat{\theta}_m + \hat{\theta}_{c1} + \hat{\phi}_2 \\ n * \cos(\hat{\theta}_{c2} + \hat{\theta}_m) - l * \cos(\hat{\theta}_m) + x_c \\ n * \sin(\hat{\theta}_{c2} + \hat{\theta}_m) - l * \sin(\hat{\theta}_m) + y_c \\ \hat{\theta}_m + \hat{\theta}_{c2} + \hat{\phi}_3 \\ -n * \cos(\hat{\theta}_{c2} + \hat{\theta}_m) - l * \cos(\hat{\theta}_m) + x_c \\ -n * \sin(\hat{\theta}_{c2} + \hat{\theta}_m) - l * \cos(\hat{\theta}_m) + x_c \\ \hat{\theta}_m + \hat{\theta}_{c2} + \hat{\phi}_4 \end{bmatrix} \quad (\text{Eq 33})$$

Table 4: Examples in the test bed folder and an explanation of their configuration

Name	Number of Robots	Configuration
Example1	2	2 robots with cluster frame defined as midpoint between both robots. The robots are limited to the plane.
Example2	4	2 robots with cluster frame defined as midpoint between both robots with the cluster frame acting as leader to another cluster frame. The follower cluster has 2 robots with its cluster frame defined as midpoint between the robots in that frame. The robots are limited to the plane.
Example3	4	A cluster of clusters. Two clusters of 2 robots each form a cluster. The robots are limited to the plane.
Example4	2	A cluster of 2 robots with one robot being located directly above the cluster frame and is free to have a different orientation than the cluster frame. The robots are limited to the plane.
Example5	2	A cluster of 2 robots with one robot being located directly above the cluster frame. This robot is also forced to be aligned with the cluster frame. This system is overconstrained. The robots are limited to the plane.
Example6	2	A cluster of 2 robots with the cluster frame arbitrarily placed. This system will have redundant degrees of freedom. The robots are limited to the plane.
Example7	2	Example 1 extended into 6 degrees of freedom per robot
Example8	3	3 robots with cluster frame defined as the average location of the three 3 robots. The robots are limited to the plane.
Example9	3	3 robots with cluster frame as the midpoint between Robot 1 and Robot 2 with Robot 3 following the cluster frame.
Example10	3	3 robots. Robot 1 is it's own cluster with the cluster frame incident with robot 1. That cluster frame plus robot 2 form another cluster frame located at the midpoint. Finally, robot 3 is following the robot 1 cluster. The robots in this example are also limited to the plane.

A full description of each cluster, including the physical view, the forward kinematics, the graph based representation, the homogeneous transform matrices used and the inverse kinematics can be found in the Appendix B.

4.4 The Velocity_Propagation_Technique Folder

The main computation of this subfolder is to execute the algorithm described in section 3.4. It does this by taking the cluster tree that was built, and traversing it accordingly. It is also capable of determining the number of degrees of freedom based on the size of the homogeneous transform

matrices, and adjusts the computation accordingly. Below is a list of all the files associated with this velocity propagation approach.

Table 5: Descriptions of the files in the Velocity_Propagation_Technique Folder

File Name	Description
<i>Invjacfxn.m</i>	The main function of the algorithm. It takes the cluster tree as an input and outputs the inverse jacobian of the system. It also saves the inverse Jacobian as an attribute to the input. The tree implementation described in section 3 made use of different color nodes. This toolbox circumvents color usage by using additional indices and counters. It also distinguishes between 3x3 and 4x4 HTMs, and calls other files as appropriate.
<i>gettheRR3.m</i>	Calculates the product of rotations for the current node. Does Step 2.2 of Section 3. Only works with 3 Degrees of Freedom (3DOF)
<i>getthev3.m</i>	Calculates the local velocity of one frame relative to its parent. Performs step 2.3
<i>rotatedwxp3.m</i>	Calculates the local angular velocity, and rotates it with respect to the Step 2.5 and 2.6
<i>getthefinv3.m</i>	Calculates the local velocity of the main cluster frame relative to global frame. This is the step 2.10 and step 2.11
<i>getthething3.m</i>	Calculates the local angular velocity. This is the same as the angular velocity propagation subroutine.
<i>gettheRR6.m</i>	The same as get the RR6, but is only called when dealing with more than 3DOF per robot.
<i>getthev6.m</i>	Does the same as getthev3, but called when user enters a 4x4 matrix.
<i>rotatedwxp6.m</i>	Does the same as rotatedwxp6, but called when user enters a 4x4 matrix.
<i>getthefinv6.m</i>	Does the same as getthefinv3, but called when user enters a 4x4 matrix.
<i>getthething6.m</i>	Does the same as getthething3, but called when user enters a 4x4 matrix.

To use the main function, one uses the command [inv_jac_prop, example3] = invjacfxn(example3). Figure 27 shows what this execution looks like in the Matlab environment.

```
>> [ inv_jac_prop, example3] = invjacfxn(example3)

inv_jac_prop =

[ 1, 0, - m*sin(tc + tc1) - l*sin(tc), cos(tc), cos(tc + tc1), 0, -m*sin(tc + tc1), 0, 0, 0, 0, 0]
[ 0, 1, m*cos(tc + tc1) + l*cos(tc), sin(tc), sin(tc + tc1), 0, m*cos(tc + tc1), 0, 0, 0, 0, 0]
[ 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0]
[ 1, 0, m*sin(tc + tc1) - l*sin(tc), cos(tc), -cos(tc + tc1), 0, m*sin(tc + tc1), 0, 0, 0, 0, 0]
[ 0, 1, l*cos(tc) - m*cos(tc + tc1), sin(tc), -sin(tc + tc1), 0, -m*cos(tc + tc1), 0, 0, 0, 0, 0]
[ 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0]
[ 1, 0, l*sin(tc) - n*sin(tc + tc2), -cos(tc), cos(tc + tc2), 0, -n*sin(tc + tc2), 0, 0, 0, 0, 0]
[ 0, 1, n*cos(tc + tc2) - l*cos(tc), -sin(tc), sin(tc + tc2), 0, n*cos(tc + tc2), 0, 0, 0, 0, 0]
[ 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0]
[ 1, 0, n*sin(tc + tc2) + l*sin(tc), -cos(tc), -cos(tc + tc2), 0, n*sin(tc + tc2), 0, 0, 0, 0, 0]
[ 0, 1, - n*cos(tc + tc2) - l*cos(tc), -sin(tc), -sin(tc + tc2), 0, -n*cos(tc + tc2), 0, 0, 0, 0, 0]
[ 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0]

example3 =

robotnodes: [4 5 6 7]
clusternodes: [1 2 3]
htm_tree: [1x1 tree]
txt_tree: [1x1 tree]
var: [1x12 sym]
inv_kin: [12x1 sym]
dirinvjac: [12x12 sym]
propinvjac: [12x12 sym]
```

Figure 27: The function *invjacfxn.m* being used on *example3* variable

Chapter 5: Results

This section will state the results of implementing this algorithm in Matlab. It will present the findings from 10 examples that the algorithm developer created, and an example from another researcher in Santa Clara University's Robotic Systems Lab. Finally, it outlines some additional testing that can be done to test the robustness of this Matlab implementation.

5.1 Results from Test_Bed Examples

Each of the ten systems in the Test_Bed Folder had the inverse Jacobian computed using the functions *dirinvjac.m* and *invjacfxn.m*. Each result for each example was compared element by element in the matrix to ensure that they are indeed the same. For all ten examples, the matrices produce matched. An example of these can be seen in Chapter 4 in figures 24 and figure 27.

An interesting finding was that by using the function which uses the newly created algorithm, a less compact form of each of the elements in the matrix were obtained. This may be the result of some optimization within Matlab's built-in 'Symbolic Math Toolbox'. However, after using simplification features, it is clear that the results were the same. An attempt was made to automate this step, but was unsuccessful.

5.2 Results from a Third Party Cluster

To remove any bias that may be present in the ten example clusters listed above, it was suggested that the algorithm be tested on a cluster definition not previously encountered by the author. Alex Mulcahy of Santa Clara University's Robotic Systems Lab requested an analysis of the following cluster given his use of this particular cluster definition in his own research.

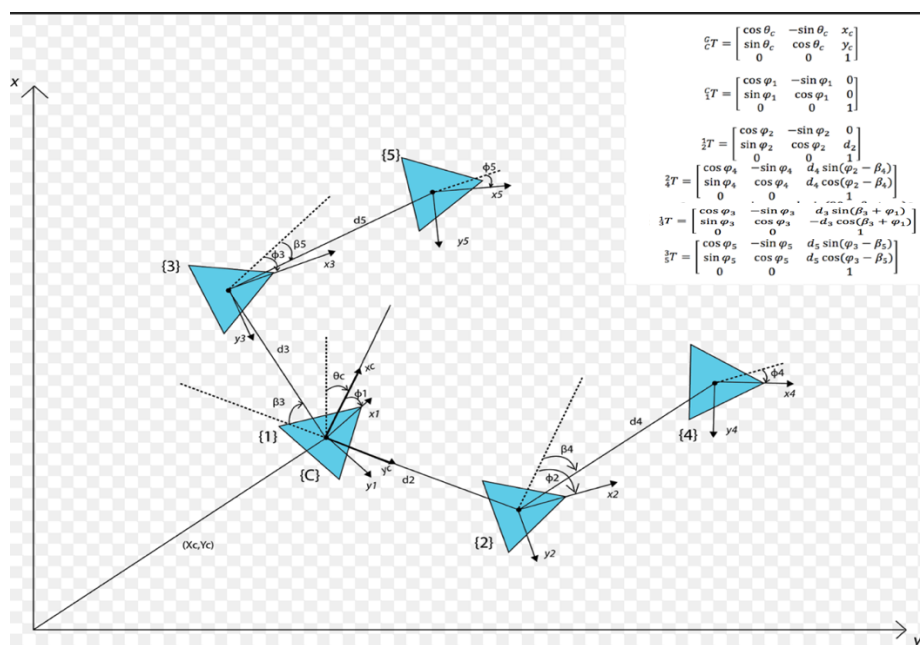


Figure 28: The Alex Cluster. A 5 robot cluster with robots 1 and 2 forming the main cluster, robot 4 following robot 2, robot 3 following robot 1, and robot 5 following robot 3.

Table 6: Other Cluster Definition to test Robustness of Toolbox

Example	Number of Robots	Configuration	Reason for Testing
Example 11	20	Each with 3 degrees of Freedom.	Will test the ability to work work with a large number of robots.
Example 12	4	Each with 6 degrees of freedom, not limited to the plane. Cluster frame at the centre of the pyramid formed.	This will fully test its ability to work with spacecraft and other 6DOF vehicles.
Example 13	4	Each with 6 degrees of freedom, tested in nonstandard axes.	Results could be used to expand the field of cluster control in a mathematical sense.
Example 14	16	A cluster of clusters of clusters.	Test ability to scale depth.

Chapter 6: Conclusions

This Chapter summarizes the work that has been done in this research. It summarizes the results and conclusions drawn, as well as outlines possibilities for future work.

6.1 Summary

This thesis has completed its main objective of automating the process of computing the inverse Jacobian. Several steps were taken in order to achieve this objective.

The frame equations that were presented by Dr. Chris Kitts were studied in depth to guarantee a throughout understanding of the mathematics involved in this computation. From there, a few test cases were computed manually to ensure proper understanding of the equations.

Once the equations were understood, substantial effort was put into finding a graph based representation for cluster space formulations. Graphs were considered as they lend themselves to existing theories from algorithmic information theory. Trees were considered particularly for clusters as it was an easy way to capture hierarchal information.

Once this graph based representation was formalized, an algorithm was developed that makes use of this representation. The algorithm, by traversing the tree and visiting relevant nodes, executed the frame propagation equations, thus allowing for the inverse Jacobian to be built.

This algorithm was then implemented in Matlab, and all the files created were saved to a Matlab toolbox. A collection of several cluster definitions were then developed to test the algorithm. This included defining the formal set of kinematic equations, depicting them as tree structures, and determining the homogeneous transform matrices needed.

The result is the initial version of a Matlab Toolbox that successfully automates the computation of the inverse Jacobian Matrix for a cluster of robots.

The results were verified by comparing the Jacobian Matrix that was obtained to one obtained by a derivation based technique. All results were mathematically equal, although certain elements in the matrix were sometimes obtained in a more compact form.

By completing this research, a systematic way for determining the inverse Jacobian of a cluster of robots can now be done quickly and systematically. This allows us to build controllers for more cluster configurations rapidly. Doing this can allow for a group of robots to be tested in simulation in different geometric descriptions, and may help a researcher decide if he wants a more or less centralized description for the group of robots being controlled.

It also expands on previous work in cluster control on Jacobian analysis, such as poor condition number avoidance. Rapid development of these matrices means that they can be studied more readily.

6.2 Future Work

Currently this toolbox only works as a proof of concept and is not optimized for run-time. The program can be optimized in Matlab, or transcribed into another language such as python for possibly faster runtimes.

Further optimization in computational efficiency can be achieved by modification of the algorithm. For example, the angular velocity propagation subroutine is already computed as part of the linear velocity propagation subroutine. Capturing that information would prevent the need to re-compute it.

The toolbox can also be developed further to provide more value to fellow researchers. Further files can be written that would automatically populate the computed inverse Jacobian into a controller in a Simulink model. It can also be modified to replace the symbols with actual values and compute a numerical value for the inverse Jacobian based on the analytical model.

Further work could also involve creating a tool to help researchers create a physical sketch of the multirobot system, and use that information to automatically create a cluster tree, as well as to determine the HTM's for each node.

The forward Jacobian is also still computed manually. Finding a faster technique for quicker computation of the forward Jacobian would directly compliment the work that has been done in this research.

Aside from graph theory, other mathematical techniques can be explored. For example, utilizing certain features of skew symmetric matrices can also be used to increase efficiency. Forcing our matrices to be skew symmetric may only work on certain clusters with restricted types of movements. This may be beneficial when small computation size required at the cost of some loss in performance.

Other techniques such as quaternions promise even faster computational efficiency, although this may require thinking of clusters in new ways and formalizing a way to think of them.

References

- [1] Farinelli, Alessandro, Luca Iocchi, and Daniele Nardi. "Multirobot systems: a classification focused on coordination." *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 34.5 (2004): 2015-2028.
- [2] Kitts, Christopher A., and Ignacio Mas. "Cluster space specification and control of mobile multirobot systems." *IEEE/ASME Transactions on Mechatronics* 14.2 (2009): 207-218.
- [3] Parker, Lynne E. "Current state of the art in distributed autonomous mobile robotics." *Distributed Autonomous Robotic Systems 4*. Springer Japan, 2000. 3-12.
- [4] Vincent, Regis, Dieter Fox, Jonathan Ko, Kurt Konolige, Benson Limketkai, Benoit Morisset, Charles Ortiz, Dirk Schulz, and Benjamin Stewart. "Distributed multirobot exploration, mapping, and task allocation." *Annals of Mathematics and Artificial Intelligence* 52.2-4 (2008): 229-255.
- [5] Anderson, Stuart O., Reid Simmons, and Dani Golberg. "Maintaining line of sight communications networks between planetary rovers." *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*. Vol. 3. IEEE, 2003: 2266-2272.
- [6] Ren, Wei, and Randal Beard. "Decentralized scheme for spacecraft formation flying via the virtual structure approach." *Journal of Guidance, Control, and Dynamics* 27.1 (2004): 73-82.
- [7] Kitts, Christopher, Thomas Adamek, Michael Vlahos, Anne Mahacek, Killian Poore, Jorge Guerra, Michael Neumann, Matthew Chin, and Mike Rasay. "An underwater robotic testbed for multi-vehicle control." *2014 IEEE/OES Autonomous Underwater Vehicles (AUV)*. IEEE, 2014: 1-8.
- [8] Michael, Nathan, Daniel Mellinger, Quentin Lindsey, and Vijay Kumar.. "The grasp multiple micro-uav testbed." *IEEE Robotics & Automation Magazine* 17.3 (2010): 56-65.
- [9] Mariottini, Gian Luca, Fabio Morbidi, Domenico Prattichizzo, Nicholas Vander Valk, Nathan Michael, George Pappas, and Kostas Daniilidis. "Vision-based localization for leader–follower formation control." *IEEE Transactions on Robotics* 25.6 (2009): 1431-1438.
- [10] Smith, Brian, Ayanna Howard, John-Michael McNew, Jiuguang Wang, and Magnus Egerstedt. "Multi-robot deployment and coordination with embedded graph grammars." *Autonomous Robots* 26.1 (2009): 79-98.
- [11] Das, Aveek K., Rafael Fierro, Vijay Kumar, James P. Ostrowski, John Spletzer, and Camillo J. Taylor. "A vision-based formation control framework." *IEEE transactions on robotics and automation* 18.5 (2002): 813-825.
- [12] Vail, Douglas, and Manuela Veloso. "Multi-robot dynamic role assignment and coordination through shared potential fields." *Multi-robot systems* (2003): 87-98.
- [13] Schneider, Frank E., and Dennis Wildermuth. "A potential field based approach to multi robot formation navigation." *Robotics, Intelligent Systems and Signal Processing, 2003. Proceedings. 2003 IEEE International Conference on*. Vol. 1. IEEE, 2003: 680-685.
- [14] Kitts, Christopher A., Kyle Stanhouse, and Piya Chindaphorn. "Cluster space collision avoidance for mobile two-robot systems." *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2009: 1941-1948.

- [15] Leonard, Naomi Ehrich, and Edward Fiorelli. "Virtual leaders, artificial potentials and coordinated control of groups." *Decision and Control, 2001. Proceedings of the 40th IEEE Conference on*. Vol. 3. IEEE, 2001:2968-2973.
- [16] Xie, Xiao-Feng, Stephen F. Smith, and Gregory J. Barlow. "Schedule-Driven Coordination for Real-Time Traffic Network Control." *ICAPS*. 2012: 323-331
- [17] Olfati-Saber, Reza. "Flocking for multi-agent dynamic systems: Algorithms and theory." *IEEE Transactions on automatic control* 51.3 (2006): 401-420.
- [18] Mas, Ignacio, and Christopher A. Kitts. "Dynamic Control of Mobile Multirobot Systems: The Cluster Space Formulation." *IEEE Access* 2 (2014): 558-570.
- [19] Ferber, Jacques. *Multi-agent systems: an introduction to distributed artificial intelligence*. Vol. 1. Reading: Addison-Wesley, 1999.
- [20] Craig, John J. *Introduction to robotics: mechanics and control*. Vol. 3. Upper Saddle River: Pearson Prentice Hall, 2005.
- [21] Tokarz, Krzysztof, and Slawosz Kieltyka. "Geometric approach to inverse kinematics for arm manipulator." *Proceedings of the 14th WSEAS international conference on Systems (ICS'10): part of the 14th WSEAS CSCC multiconference*. Vol. 2. 2010: 682-687.
- [22] Lilly, Kathryn W., and David E. Orin. "Alternate formulations for the manipulator inertia matrix." *The International Journal of Robotics Research* 10.1 (1991): 64-74.
- [23] Wang, Xuguang, and Jean Pierre Verriest. "A geometric algorithm to predict the arm reach posture for computer-aided ergonomic evaluation." *The journal of visualization and computer animation* 9.1 (1998): 33-47.
- [24] Raghavan, Madhusudan, and Bernard Roth. "Kinematic analysis of the 6R manipulator of general geometry." *Proc. Fifth Int. Symposium on Robotics Research, MIT Press, Cambridge*. 1990:1-28.
- [25] Corke, Peter. *Robotics, vision and control: fundamental algorithms in MATLAB*. Vol. 73. Springer, 2011.
- [26] Tsai, Lung-Wen. *Robot analysis: the mechanics of serial and parallel manipulators*. John Wiley & Sons, 1999.
- [27] Weisstein, Eric W. "Jacobian." From *MathWorld*--A Wolfram Web Resource. <http://mathworld.wolfram.com/Jacobian.html>
- [28] Boullion, Thomas L., and Patrick L. Odell. "Generalized inverse matrices." (1971).
- [29] Orin, David E., and William W. Schrader. "Efficient computation of the Jacobian for robot manipulators." *The International Journal of Robotics Research* 3.4 (1984): 66-75.
- [30] Mas, Ignacio. *Cluster space framework for multi-robot formation control*. Diss. Santa Clara University, 2011.

Appendix A – Examples used in the Test_Bed Folder

Example 1: A Cluster of Two Robots

Physical Diagram:

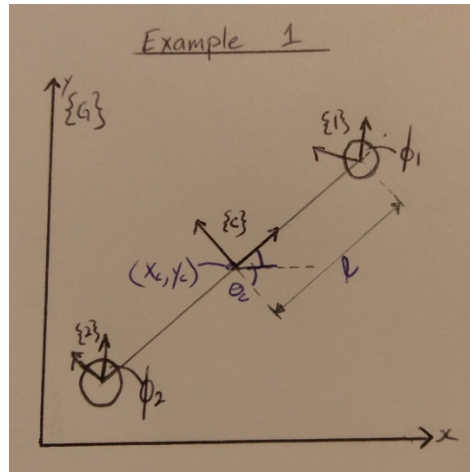


Figure 31: A Physical view of a cluster of 2 robots for example1

Forward Kinematics:

$$\begin{bmatrix} \hat{x}_c \\ \hat{y}_c \\ \hat{\phi}_1 \\ \hat{\phi}_2 \\ \hat{\theta}_c \\ \hat{l} \end{bmatrix} = \begin{bmatrix} \frac{(x_1 + x_2)}{2} \\ \frac{(y_1 + y_2)}{2} \\ \hat{\theta}_1 - \theta_c \\ \hat{\theta}_2 - \theta_c \\ \text{atan2}((y_2 - y_1), (x_2 - x_1)) \\ \frac{1}{2} \sqrt{(y_1 - y_2)^2 + (x_2 - x_1)^2} \end{bmatrix}$$

Tree Structure:

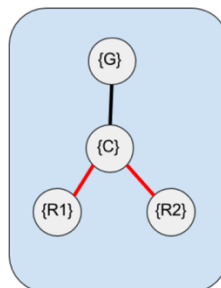


Figure 32: A Graph representing a cluster of 2 robots for example1

Transformation Matrices:

$${}^cT = \begin{bmatrix} \cos(\hat{\theta}_c) & -\sin(\hat{\theta}_c) & \hat{x}_c \\ \sin(\hat{\theta}_c) & \cos(\hat{\theta}_c) & \hat{y}_c \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^c_1T = \begin{bmatrix} \cos(\hat{\phi}_1) & -\sin(\hat{\phi}_1) & l \\ \sin(\hat{\phi}_1) & \cos(\hat{\phi}_1) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^c_2T = \begin{bmatrix} \cos(\hat{\phi}_2) & -\sin(\hat{\phi}_2) & -l \\ \sin(\hat{\phi}_2) & \cos(\hat{\phi}_2) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^c_1T = {}^cT * {}^c_1T = \begin{bmatrix} \cos(\hat{\theta}_c + \hat{\phi}_1) & -\sin(\hat{\theta}_c + \hat{\phi}_1) & l * \cos(\hat{\theta}_c) + \hat{x}_c \\ \sin(\hat{\theta}_c + \hat{\phi}_1) & \cos(\hat{\theta}_c + \hat{\phi}_1) & l * \sin(\hat{\theta}_c) + \hat{y}_c \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^c_2T = {}^cT * {}^c_2T = \begin{bmatrix} \cos(\hat{\theta}_c + \hat{\phi}_2) & -\sin(\hat{\theta}_c + \hat{\phi}_2) & -l * \cos(\hat{\theta}_c) + \hat{x}_c \\ \sin(\hat{\theta}_c + \hat{\phi}_2) & \cos(\hat{\theta}_c + \hat{\phi}_2) & -l * \sin(\hat{\theta}_c) + \hat{y}_c \\ 0 & 0 & 1 \end{bmatrix}$$

Inverse Kinematics

$$\begin{bmatrix} x_1 \\ y_1 \\ \hat{\theta}_1 \\ x_2 \\ y_2 \\ \hat{\theta}_2 \end{bmatrix} = \begin{bmatrix} l * \cos(\hat{\theta}_c) + \hat{x}_c \\ l * \sin(\hat{\theta}_c) + \hat{y}_c \\ \hat{\theta}_c + \hat{\phi}_1 \\ -l * \cos(\hat{\theta}_c) + \hat{x}_c \\ -l * \sin(\hat{\theta}_c) + \hat{y}_c \\ \hat{\theta}_c + \hat{\phi}_2 \end{bmatrix}$$

Example 2: Two Clusters of Two Robots in a Leader Follower Formation

Physical Diagram

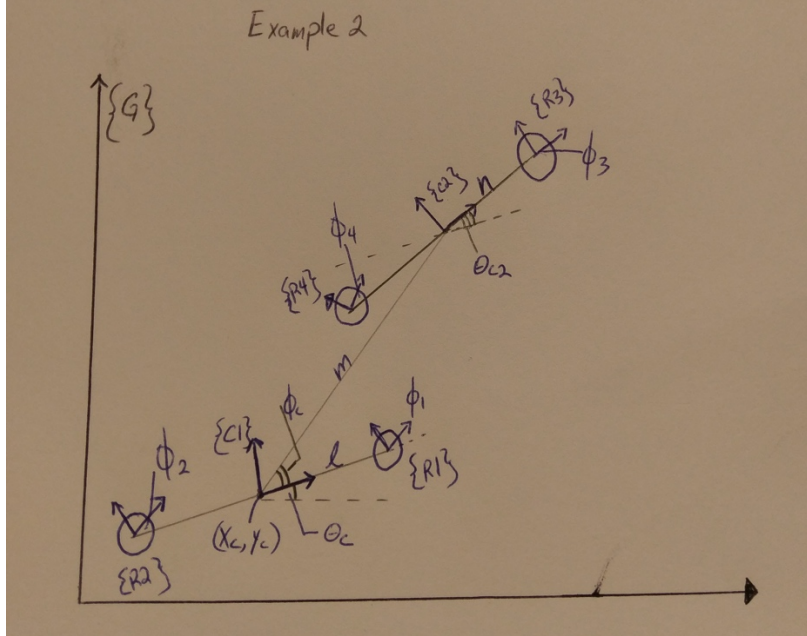


Figure 33: Physical View of 2 clusters of 2 robots each in a leader follower configuration, where cluster {C1} is leader and {C2} is follower.

Forward Kinematics

$$\begin{bmatrix} \hat{x}_c \\ \hat{y}_m \\ \hat{\phi}_1 \\ \hat{\phi}_2 \\ \hat{\phi}_3 \\ \hat{\phi}_4 \\ \hat{\theta}_{c1} \\ \hat{l} \\ \hat{m} \\ \hat{n} \\ \hat{\theta}_{c2} \\ \hat{\phi}_c \end{bmatrix} = \begin{bmatrix} \frac{(x_1 + x_2)}{2} \\ \frac{(y_1 + y_2)}{2} \\ \hat{\theta}_1 - \hat{\theta}_{c1} \\ \hat{\theta}_2 - \hat{\theta}_{c1} \\ \hat{\theta}_3 - \theta_{c1} - \hat{\phi}_c \\ \hat{\theta}_4 - \hat{\theta}_{c1} - \hat{\phi}_c \\ \text{atan2}((y_2 - y_1), (x_2 - x_1)) \\ \frac{1}{2} \sqrt{(y_1 - y_2)^2 + (x_2 - x_1)^2} \\ \sqrt{\left(\frac{(x_1 + x_2)}{2} - \frac{(x_3 + x_4)}{2}\right)^2 + \left(\frac{(y_1 + y_2)}{2} - \frac{(y_3 + y_4)}{2}\right)^2} \\ \frac{1}{2} \sqrt{(y_4 - y_3)^2 + (x_4 - x_3)^2} \\ \text{acos}\left(\frac{((y_3 - y_{c1})^2 + (x_3 - x_{c1})^2 - 4\hat{l}^2 - \hat{m}^2)}{4ln}\right) - \hat{\phi}_c + \pi \\ \text{acos}\left(\frac{((y_{c2} - y_1)^2 + (x_{c2} - x_1)^2 - 4\hat{l}^2 - \hat{m}^2)}{4lm}\right) \end{bmatrix}$$

Tree Diagram

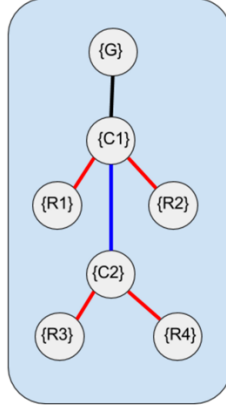


Figure 34: A Graph representing 2 clusters of 2 robots each in a leader follower configuration, where cluster {C1} is leader and {C2} is follower.

Transformation Matrices:

$${}^G c_1 T = \begin{bmatrix} \cos(\hat{\theta}_{c1}) & -\sin(\hat{\theta}_{c1}) & \hat{x}_c \\ \sin(\hat{\theta}_{c1}) & \cos(\hat{\theta}_{c1}) & \hat{y}_c \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^{c_1} T_1 = \begin{bmatrix} \cos(\hat{\phi}_1) & -\sin(\hat{\phi}_1) & l \\ \sin(\hat{\phi}_1) & \cos(\hat{\phi}_1) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^{c_1} T_2 = \begin{bmatrix} \cos(\hat{\phi}_2) & -\sin(\hat{\phi}_2) & -l \\ \sin(\hat{\phi}_2) & \cos(\hat{\phi}_2) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^{c_2} T = \begin{bmatrix} \cos(\hat{\theta}_{c2}) & -\sin(\hat{\theta}_{c2}) & m * \cos(\hat{\phi}_c) \\ \sin(\hat{\theta}_{c2}) & \cos(\hat{\theta}_{c2}) & m * \sin(\hat{\phi}_c) \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^{c_2} T_3 = \begin{bmatrix} \cos(\hat{\phi}_3) & -\sin(\hat{\phi}_3) & n \\ \sin(\hat{\phi}_3) & \cos(\hat{\phi}_3) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^{c_2} T_4 = \begin{bmatrix} \cos(\hat{\phi}_4) & -\sin(\hat{\phi}_4) & -n \\ \sin(\hat{\phi}_4) & \cos(\hat{\phi}_4) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Inverse Kinematics:

$$\begin{bmatrix} x_1 \\ y_1 \\ \hat{\theta}_1 \\ x_2 \\ y_2 \\ \hat{\theta}_2 \\ x_3 \\ y_3 \\ \hat{\theta}_3 \\ x_4 \\ y_4 \\ \hat{\theta}_4 \end{bmatrix} = \begin{bmatrix} l * \cos(\hat{\theta}_{c1} + \hat{\phi}_c) + \hat{x}_c \\ l * \sin(\hat{\theta}_{c1} + \hat{\phi}_c) + \hat{y}_c \\ \hat{\theta}_{c1} + \hat{\phi}_1 \\ -l * \cos(\hat{\theta}_{c1} + \hat{\phi}_c) + \hat{x}_c \\ -l * \sin(\hat{\theta}_{c1} + \hat{\phi}_c) + \hat{y}_c \\ \hat{\theta}_{c1} + \hat{\phi}_2 \\ m * \cos(\hat{\phi}_c + \hat{\theta}_{c1}) + n * \cos(\hat{\theta}_{c2} + \hat{\theta}_{c1}) + \hat{x}_c \\ m * \cos(\hat{\phi}_c + \hat{\theta}_{c1}) + n * \sin(\hat{\theta}_{c2} + \hat{\theta}_{c1}) + \hat{y}_c \\ \hat{\theta}_c + \hat{\phi}_c + \hat{\phi}_3 \\ m * \cos(\hat{\phi}_c + \hat{\theta}_{c1}) - n * \cos(\hat{\theta}_{c2} + \hat{\theta}_{c1}) + \hat{x}_c \\ m * \cos(\hat{\phi}_c + \hat{\theta}_{c1}) - n * \sin(\hat{\theta}_{c2} + \hat{\theta}_{c1}) + \hat{y}_c \\ \hat{\theta}_c + \hat{\phi}_c + \hat{\phi}_4 \end{bmatrix}$$

Example 3: A Cluster of two 2-robot clusters

Physical Diagram

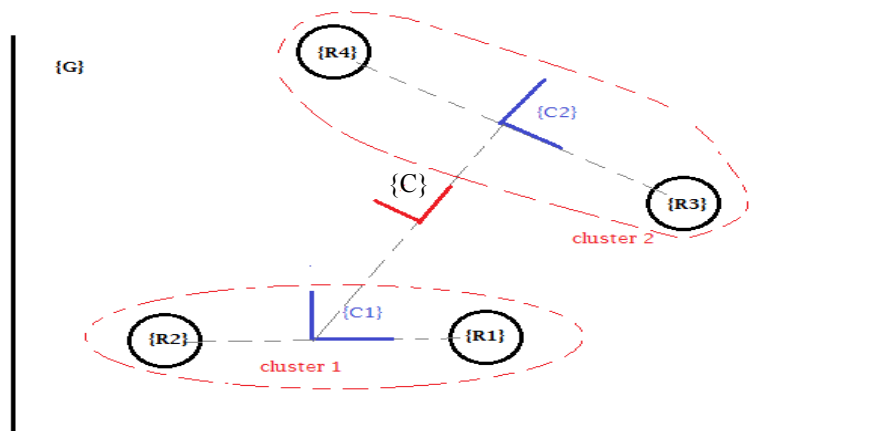


Figure 35: A Physical View of Example3, a cluster of two 2-robot clusters.

Forward Kinematics

$$\begin{bmatrix} \hat{x}_c \\ \hat{y}_m \\ \hat{\phi}_1 \\ \hat{\phi}_2 \\ \hat{\phi}_3 \\ \hat{\phi}_4 \\ \hat{\theta}_c \\ \hat{l} \\ \hat{m} \\ \hat{n} \\ \hat{\theta}_{c1} \\ \hat{\theta}_{c2} \end{bmatrix} = \begin{bmatrix} \frac{(x_1 + x_2 + x_3 + x_4)}{4} \\ \frac{(y_1 + y_2 + y_3 + y_4)}{4} \\ \hat{\theta}_1 - \hat{\theta}_{c1} - \theta_c \\ \hat{\theta}_2 - \hat{\theta}_{c1} - \hat{\theta}_c \\ \hat{\theta}_3 - \theta_{c2} - \theta_c \\ \hat{\theta}_4 - \hat{\theta}_{c2} - \hat{\theta}_c \\ \text{atan2}((y_m - y_{c1}), (x_m - \frac{(x_1 + x_2)}{2})) \\ \sqrt{\left(\frac{(y_1 + y_2)}{2} - y_m\right)^2 + \left(\frac{(x_1 + x_2)}{2} - x_m\right)^2} \\ \sqrt{\left(\frac{(y_1 + y_2)}{2} - y_1\right)^2 + \left(\frac{(x_1 + x_2)}{2} - x_1\right)^2} \\ \sqrt{\left(\frac{(y_3 + y_4)}{2} - y_1\right)^2 + \left(\frac{(x_3 + x_4)}{2} - x_1\right)^2} \\ \text{acos}\left(\frac{((y_m - y_1)^2 + (x_m - x_1)^2 - \hat{l}^2 - \hat{m}^2)}{4lm}\right) \\ \text{acos}\left(\frac{((y_m - y_3)^2 + (x_m - x_3)^2 - \hat{l}^2 - \hat{n}^2)}{4ln}\right) \end{bmatrix}$$

Tree Diagram

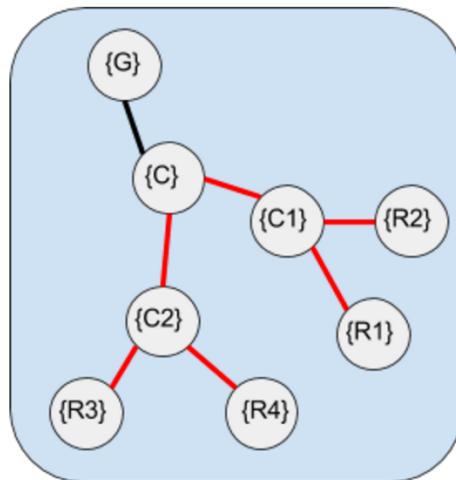


Figure 36: A Graph representing a cluster of clusters. Here cluster {C1} and {C2} form a single cluster {C}.

Transformation Matrices

$${}^G_c T = \begin{bmatrix} \cos(\hat{\theta}_c) & -\sin(\hat{\theta}_c) & x_c \\ \sin(\hat{\theta}_c) & \cos(\hat{\theta}_c) & y_c \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^c_{c1} T = \begin{bmatrix} \cos(\hat{\theta}_{c1}) & -\sin(\hat{\theta}_{c1}) & l \\ \sin(\hat{\theta}_{c1}) & \cos(\hat{\theta}_{c1}) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^c_{c2} T = \begin{bmatrix} \cos(\hat{\theta}_{c2}) & -\sin(\hat{\theta}_{c2}) & -l \\ \sin(\hat{\theta}_{c2}) & \cos(\hat{\theta}_{c2}) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^c_{R1} T = \begin{bmatrix} \cos(\hat{\phi}_1) & -\sin(\hat{\phi}_1) & m \\ \sin(\hat{\phi}_1) & \cos(\hat{\phi}_1) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^c_{R2} T = \begin{bmatrix} \cos(\hat{\phi}_2) & -\sin(\hat{\phi}_2) & -m \\ \sin(\hat{\phi}_2) & \cos(\hat{\phi}_2) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^c_{R3} T = \begin{bmatrix} \cos(\hat{\phi}_3) & -\sin(\hat{\phi}_3) & n \\ \sin(\hat{\phi}_3) & \cos(\hat{\phi}_3) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^c_{R4} T = \begin{bmatrix} \cos(\hat{\phi}_4) & -\sin(\hat{\phi}_4) & -n \\ \sin(\hat{\phi}_4) & \cos(\hat{\phi}_4) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Inverse Kinematics

$$\begin{bmatrix} x_1 \\ y_1 \\ \hat{\theta}_1 \\ x_2 \\ y_2 \\ \hat{\theta}_2 \\ x_3 \\ y_3 \\ \hat{\theta}_3 \\ x_4 \\ y_4 \\ \hat{\theta}_4 \end{bmatrix} = \begin{bmatrix} m * \cos(\hat{\theta}_{c1} + \hat{\theta}_c) + l * \cos(\hat{\theta}_c) + x_c \\ m * \sin(\hat{\theta}_{c1} + \hat{\theta}_c) + l * \sin(\hat{\theta}_c) + y_c \\ \hat{\theta}_c + \hat{\theta}_{c1} + \hat{\phi}_1 \\ -m * \cos(\hat{\theta}_{c1} + \hat{\theta}_c) + l * \cos(\hat{\theta}_c) + x_c \\ -m * \sin(\hat{\theta}_{c1} + \hat{\theta}_c) + l * \sin(\hat{\theta}_c) + y_c \\ \hat{\theta}_c + \hat{\theta}_{c1} + \hat{\phi}_2 \\ n * \cos(\hat{\theta}_{c2} + \hat{\theta}_c) - l * \cos(\hat{\theta}_c) + x_c \\ n * \sin(\hat{\theta}_{c2} + \hat{\theta}_c) - l * \sin(\hat{\theta}_c) + y_c \\ \hat{\theta}_c + \hat{\theta}_{c2} + \hat{\phi}_3 \\ -n * \cos(\hat{\theta}_{c2} + \hat{\theta}_c) - l * \cos(\hat{\theta}_c) + x_c \\ -n * \sin(\hat{\theta}_{c2} + \hat{\theta}_c) - l * \cos(\hat{\theta}_c) + x_c \\ \hat{\theta}_c + \hat{\theta}_{c2} + \hat{\phi}_4 \end{bmatrix}$$

Example 4: A Two Robot Cluster with the Cluster frame Position Determined by One Robot

Physical View

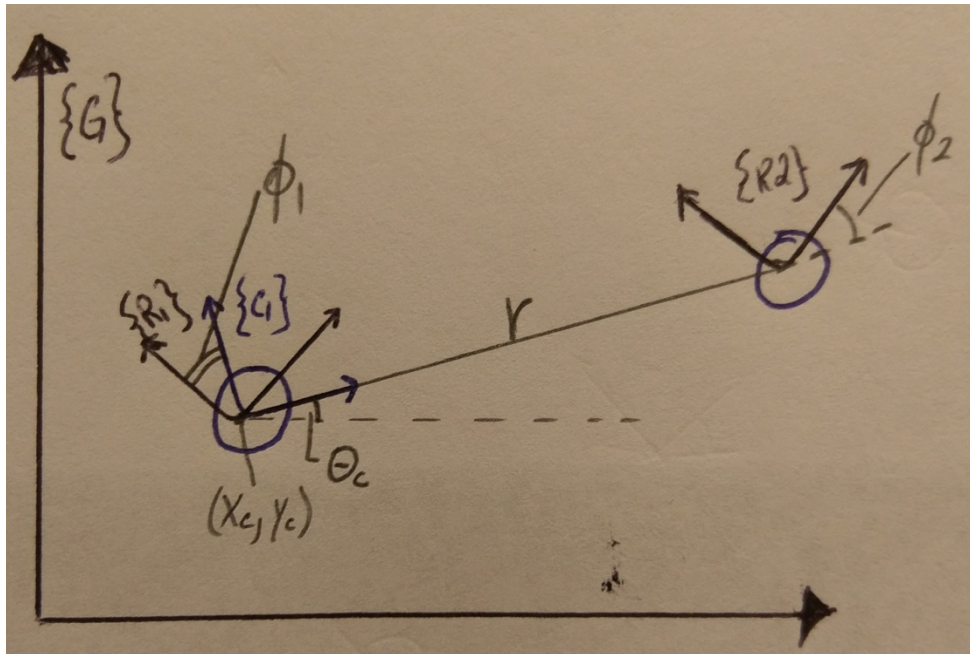


Figure 37: A Two robot cluster sharing a single cluster frame. The Cluster frame is located on top of Robot 1. However, robot 1 is free to have a different orientation than the cluster frame.

Forward Kinematics

$$\begin{bmatrix} \hat{x}_c \\ \hat{y}_c \\ \hat{\phi}_1 \\ \hat{\phi}_2 \\ \hat{\theta}_c \\ \hat{r} \end{bmatrix} = \begin{bmatrix} \hat{x}_1 \\ \hat{y}_1 \\ \hat{\theta}_1 - \theta_c \\ \hat{\theta}_2 - \hat{\theta}_c \\ \text{atan2}((y_2 - y_1), (x_2 - x_1)) \\ \sqrt{(y_1 - y_2)^2 + (x_2 - x_1)^2} \end{bmatrix}$$

Tree Diagram

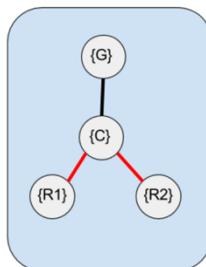


Figure 38: Tree Diagram of a Two robot cluster sharing a single cluster frame. The Cluster frame is located on top of Robot 1.

Transformation Matrices

$${}^cT = \begin{bmatrix} \cos(\hat{\theta}_c) & -\sin(\hat{\theta}_c) & \hat{x}_c \\ \sin(\hat{\theta}_c) & \cos(\hat{\theta}_c) & \hat{y}_c \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^cT_1 = \begin{bmatrix} \cos(\hat{\phi}_1) & -\sin(\hat{\phi}_1) & 0 \\ \sin(\hat{\phi}_1) & \cos(\hat{\phi}_1) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^cT_2 = \begin{bmatrix} \cos(\hat{\phi}_2) & -\sin(\hat{\phi}_2) & r \\ \sin(\hat{\phi}_2) & \cos(\hat{\phi}_2) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^cT_1 = {}^cT * {}^cT_1 = \begin{bmatrix} \cos(\hat{\theta}_c + \hat{\phi}_1) & -\sin(\hat{\theta}_c + \hat{\phi}_1) & \hat{x}_c \\ \sin(\hat{\theta}_c + \hat{\phi}_1) & \cos(\hat{\theta}_c + \hat{\phi}_1) & \hat{y}_c \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^cT_2 = {}^cT * {}^cT_2 = \begin{bmatrix} \cos(\hat{\theta}_c + \hat{\phi}_2) & -\sin(\hat{\theta}_c + \hat{\phi}_2) & r * \cos(\hat{\theta}_c) + \hat{x}_c \\ \sin(\hat{\theta}_c + \hat{\phi}_2) & \cos(\hat{\theta}_c + \hat{\phi}_2) & r * \sin(\hat{\theta}_c) + \hat{y}_c \\ 0 & 0 & 1 \end{bmatrix}$$

Inverse Kinematics:

$$\begin{bmatrix} x_1 \\ y_1 \\ \hat{\theta}_1 \\ x_2 \\ y_2 \\ \hat{\theta}_2 \end{bmatrix} = \begin{bmatrix} \hat{x}_c \\ \hat{y}_c \\ \hat{\theta}_c + \hat{\phi}_1 \\ r * \cos(\hat{\theta}_c) + \hat{x}_c \\ r * \sin(\hat{\theta}_c) + \hat{y}_c \\ \hat{\theta}_c + \hat{\phi}_2 \end{bmatrix}$$

Example 5: Two Robot Formation with Cluster frame incident to Robot 1

Physical Diagram

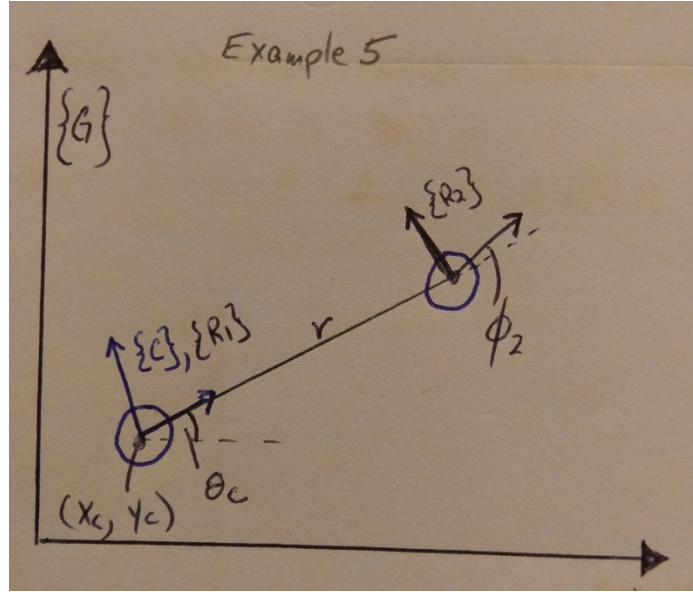


Figure 39: Physical view diagram of a Two robot cluster sharing a single cluster frame. The Cluster frame is located on top of and oriented with Robot 1.

Forward Kinematics

$$\begin{bmatrix} \hat{x}_c \\ \hat{y}_c \\ \hat{\phi}_1 \\ \hat{\phi}_2 \\ \hat{\theta}_c \\ \hat{r} \end{bmatrix} = \begin{bmatrix} \hat{x}_1 \\ \hat{y}_1 \\ 0 \\ \hat{\theta}_2 - \hat{\theta}_c \\ \hat{\theta}_1 \\ \sqrt{(y_1 - y_2)^2 + (x_2 - x_1)^2} \end{bmatrix}$$

Tree Structure

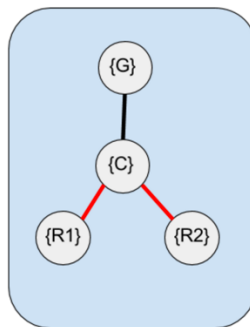


Figure 40: Tree diagram of a Two robot cluster sharing a single cluster frame. The Cluster frame is located on top of and oriented with Robot 1.

Transformation Matrices

$${}^G_cT = \begin{bmatrix} \cos(\hat{\theta}_c) & -\sin(\hat{\theta}_c) & \hat{x}_c \\ \sin(\hat{\theta}_c) & \cos(\hat{\theta}_c) & \hat{y}_c \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^c_1T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^c_2T = \begin{bmatrix} \cos(\hat{\phi}_2) & -\sin(\hat{\phi}_2) & r \\ \sin(\hat{\phi}_2) & \cos(\hat{\phi}_2) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^G_1T = {}^G_cT * {}^c_1T = \begin{bmatrix} \cos(\hat{\theta}_c) & -\sin(\hat{\theta}_c) & \hat{x}_c \\ \sin(\hat{\theta}_c) & \cos(\hat{\theta}_c) & \hat{y}_c \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^G_2T = {}^G_cT * {}^c_2T = \begin{bmatrix} \cos(\hat{\theta}_c + \hat{\phi}_2) & -\sin(\hat{\theta}_c + \hat{\phi}_2) & r * \cos(\hat{\theta}_c) + \hat{x}_c \\ \sin(\hat{\theta}_c + \hat{\phi}_2) & \cos(\hat{\theta}_c + \hat{\phi}_2) & r * \sin(\hat{\theta}_c) + \hat{y}_c \\ 0 & 0 & 1 \end{bmatrix}$$

Inverse Kinematics:

$$\begin{bmatrix} x_1 \\ y_1 \\ \hat{\theta}_1 \\ x_2 \\ y_2 \\ \hat{\theta}_2 \end{bmatrix} = \begin{bmatrix} \hat{x}_c \\ \hat{y}_c \\ \hat{\theta}_c \\ r * \cos(\hat{\theta}_c) + \hat{x}_c \\ r * \sin(\hat{\theta}_c) + \hat{y}_c \\ \hat{\theta}_c + \hat{\phi}_2 \end{bmatrix}$$

Example 6: A Two robot cluster with a redundant variable

Physical Diagram:

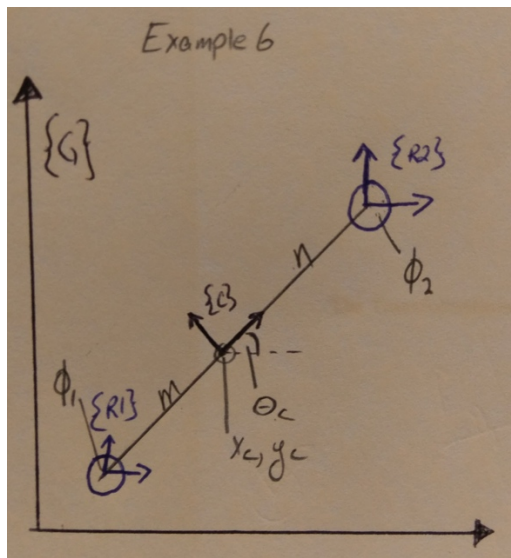


Figure 41: A Graph representing a cluster of 2 robots.

Forward Kinematics

$$\begin{bmatrix} \hat{x}_c \\ \hat{y}_c \\ \hat{\phi}_1 \\ \hat{\phi}_2 \\ \hat{\theta}_c \\ \hat{m} \\ \hat{n} \\ \hat{l} \end{bmatrix} = \begin{bmatrix} \frac{(x_1 + x_2)}{2} \\ \frac{(y_1 + y_2)}{2} \\ \hat{\theta}_1 - \theta_c \\ \hat{\theta}_2 - \hat{\theta}_c \\ \text{atan2}((y_2 - y_1), (x_2 - x_1)) \\ \frac{\sqrt{(y_1 - y_c)^2 + (x_c - x_1)^2}}{\sqrt{(y_c - y_2)^2 + (x_2 - x_c)^2}} \\ \frac{\sqrt{(y_1 - y_2)^2 + (x_2 - x_1)^2}}{\sqrt{(y_1 - y_2)^2 + (x_2 - x_1)^2}} \end{bmatrix}$$

Tree Structure

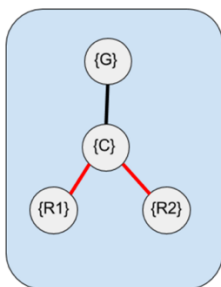


Figure 42: A Graph representing a cluster of 2 robots.

Transformation Matrices

$${}^c_2T = \begin{bmatrix} \cos(\hat{\theta}_c) & -\sin(\hat{\theta}_c) & \hat{x}_c \\ \sin(\hat{\theta}_c) & \cos(\hat{\theta}_c) & \hat{y}_c \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^c_1T = \begin{bmatrix} \cos(\hat{\phi}_1) & -\sin(\hat{\phi}_1) & m \\ \sin(\hat{\phi}_1) & \cos(\hat{\phi}_1) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^c_2T = \begin{bmatrix} \cos(\hat{\phi}_2) & -\sin(\hat{\phi}_2) & n \\ \sin(\hat{\phi}_2) & \cos(\hat{\phi}_2) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Inverse Kinematics

$$\begin{bmatrix} x_1 \\ y_1 \\ \hat{\theta}_1 \\ x_2 \\ y_2 \\ \hat{\theta}_2 \end{bmatrix} = \begin{bmatrix} m * \cos(\hat{\theta}_c) + \hat{x}_c \\ m * \sin(\hat{\theta}_c) + \hat{y}_c \\ \hat{\theta}_c + \hat{\phi}_1 \\ n * \cos(\hat{\theta}_c) + \hat{x}_c \\ n * \sin(\hat{\theta}_c) + \hat{y}_c \\ \hat{\theta}_c + \hat{\phi}_2 \end{bmatrix}$$

Example 7: Two Robot Cluster with extended dimensional degrees of Freedom

Physical Diagram

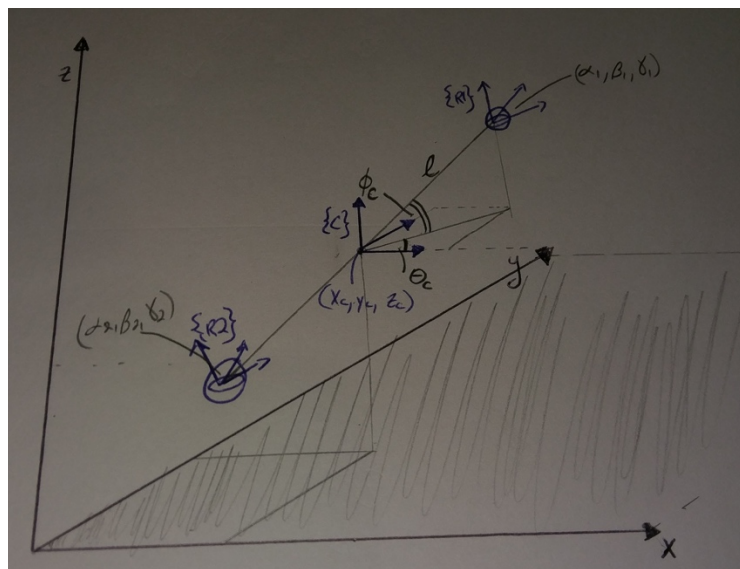


Figure 43: A sketch of a cluster of 2 robots in 3D.

Forward Kinematics

$$\begin{bmatrix} \hat{x}_c \\ \hat{y}_c \\ \hat{z}_c \\ \alpha_1 \\ \beta_1 \\ \gamma_1 \\ \alpha_2 \\ \beta_2 \\ \gamma_2 \\ \hat{\phi}_c \\ \hat{\theta}_c \\ \hat{l} \end{bmatrix} = \begin{bmatrix} \frac{(x_1 + x_2)}{2} \\ \frac{(y_1 + y_2)}{2} \\ \frac{(z_1 + z_2)}{2} \\ \hat{\alpha}_{R1} \\ \hat{\phi}_{R1} \\ \hat{\theta}_{R1} \\ \hat{\alpha}_{R2} \\ \hat{\phi}_{R2} \\ \hat{\theta}_{R2} \\ \frac{\text{atan2}((z_2 - z_1), \sqrt{(y_1 - y_2)^2 + (x_2 - x_1)^2})}{\sqrt{(z_1 - z_2)^2 + (y_1 - y_2)^2 + (x_2 - x_1)^2}} \\ \frac{\text{atan2}((y_2 - y_1), (x_2 - x_1))}{\sqrt{(z_1 - z_2)^2 + (y_1 - y_2)^2 + (x_2 - x_1)^2}} \end{bmatrix}$$

Tree Diagram

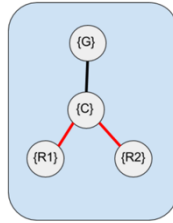


Figure 44: Two Robots in 3 Dimensional Space

Transformation Matrices

The Transformations for this system are:

$${}^G T_C = \begin{bmatrix} 1 & 0 & 0 & x_c \\ 0 & 1 & 0 & y_c \\ 0 & 0 & 1 & z_c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^C T_1 = \begin{bmatrix} \cos(\gamma_1)\cos(\beta_1) & -\cos(\gamma_1)\sin(\beta_1)\sin(\alpha_1) - \sin(\gamma_1)\cos(\alpha_1) & -\cos(\alpha_1)\sin(\beta_1)\cos(\gamma_1) + \sin(\alpha_1)\sin(\gamma_1) & l * \cos(\hat{\theta}_c)\cos(\hat{\phi}_c) \\ \sin(\gamma_1)\cos(\beta_1) & -\sin(\gamma_1)\sin(\beta_1)\sin(\alpha_1) + \cos(\gamma_1)\cos(\alpha_1) & -\cos(\alpha_1)\sin(\beta_1)\sin(\gamma_1) - \sin(\alpha_1)\cos(\gamma_1) & l * \sin(\hat{\theta}_c)\cos(\hat{\phi}_c) \\ -\sin(\beta_1) & \cos(\beta_1)\sin(\alpha_1) & \cos(\beta_1)\cos(\alpha_1) & l * \sin(\hat{\phi}_c) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^C T_2 = \begin{bmatrix} \cos(\gamma_1)\cos(\beta_1) & -\cos(\gamma_1)\sin(\beta_1)\sin(\alpha_1) - \sin(\gamma_1)\cos(\alpha_1) & -\cos(\alpha_1)\sin(\beta_1)\cos(\gamma_1) + \sin(\alpha_1)\sin(\gamma_1) & -l * \cos(\hat{\theta}_c)\cos(\hat{\phi}_c) \\ \sin(\gamma_1)\cos(\beta_1) & -\sin(\gamma_1)\sin(\beta_1)\sin(\alpha_1) + \cos(\gamma_1)\cos(\alpha_1) & -\cos(\alpha_1)\sin(\beta_1)\sin(\gamma_1) - \sin(\alpha_1)\cos(\gamma_1) & -l * \sin(\hat{\theta}_c)\cos(\hat{\phi}_c) \\ -\sin(\beta_1) & \cos(\beta_1)\sin(\alpha_1) & \cos(\beta_1)\cos(\alpha_1) & -l * \sin(\hat{\phi}_c) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Inverse Kinematics

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ \hat{\theta}_{R1} \\ \hat{\phi}_{R1} \\ \hat{\alpha}_{R1} \\ x_2 \\ y_2 \\ z_2 \\ \hat{\theta}_{R2} \\ \hat{\phi}_{R2} \\ \hat{\alpha}_{R2} \end{bmatrix} = \begin{bmatrix} x_c + l * \cos(\hat{\theta}_c) \cos(\hat{\phi}_c) \\ y_c + l * \sin(\hat{\phi}_c) \cos(\hat{\theta}_c) \\ z_c + l * \sin(\hat{\phi}_c) \\ \gamma_1 \\ \beta_1 \\ \hat{\alpha}_1 \\ x_c - l * \cos(\hat{\theta}_c) \cos(\hat{\phi}_c) \\ y_c - l * \sin(\hat{\phi}_c) \cos(\hat{\theta}_c) \\ z_c - l * \sin(\hat{\phi}_c) \\ \gamma_2 \\ \beta_2 \\ \hat{\alpha}_2 \end{bmatrix}$$

Example 8: Three Robot Cluster with Cluster Frame at Centroid Location

Physical View

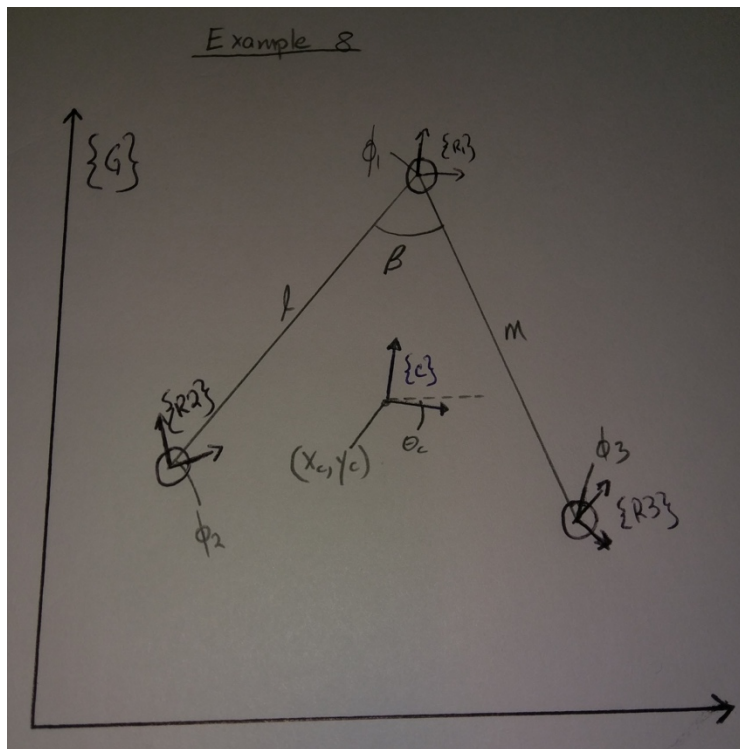


Figure 45: Physical view of a three robot Cluster in formation

Forward Kinematics

$$\begin{bmatrix} x_c \\ y_c \\ \hat{\theta}_c \\ \hat{l} \\ \hat{m} \\ \hat{\phi}_c \\ \hat{\phi}_1 \\ \hat{\phi}_2 \\ \hat{\phi}_3 \\ r \\ x \\ \hat{a} \\ \hat{b} \\ \hat{R} \end{bmatrix} = \begin{bmatrix} \frac{(x_1 + x_2 + x_3)}{3} \\ \frac{(y_1 + y_2 + y_3)}{3} \\ \text{atan2}((y_c - y_1), (x_c - x_1)) + \pi/2 \\ \frac{\sqrt{((y_2 - y_1)^2 + (x_2 - x_1)^2)}}{\sqrt{((y_3 - y_1)^2 + (x_3 - x_1)^2)}} \\ \text{acos}\left(\frac{(\hat{l}^2 + \hat{m}^2 - ((y_2 - y_3)^2 - (x_2 - x_3)^2))}{2lm}\right) \\ \frac{\hat{\theta}_1 - \hat{\theta}_c}{\hat{\theta}_2 - \hat{\theta}_c} \\ \frac{\hat{\theta}_3 - \hat{\theta}_c}{\sqrt{l^2 + m^2 - 2lm\cos(\hat{\phi}_c)}} \\ \sqrt{l^2 + \frac{r^2}{4} - 2lr\cos(\hat{a})} \\ \frac{\text{acos}(r^2 + l^2 - m^2)}{2rl} \\ \text{acos}\left(l^2 + x^2 - \frac{r^2}{4}\right) \\ \frac{2lx}{\hat{\phi}_c - \hat{a}} \\ \frac{2\hat{x}}{3} \end{bmatrix}$$

Tree Structure

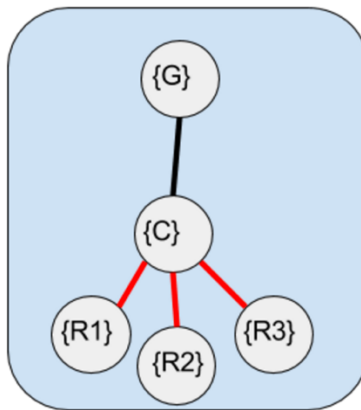


Figure 46: Cluster tree of a three robot Cluster.

Transformation Matrices

$${}^G_c T = \begin{bmatrix} \cos(\hat{\theta}_c) & -\sin(\hat{\theta}_c) & \hat{x}_c \\ \sin(\hat{\theta}_c) & \cos(\hat{\theta}_c) & \hat{y}_c \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^c_1 T = \begin{bmatrix} \cos(\hat{\phi}_1) & -\sin(\hat{\phi}_1) & 0 \\ \sin(\hat{\phi}_1) & \cos(\hat{\phi}_1) & R \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^c_2 T = \begin{bmatrix} \cos(\hat{\phi}_2) & -\sin(\hat{\phi}_2) & -l * \sin(\hat{a}) \\ \sin(\hat{\phi}_2) & \cos(\hat{\phi}_2) & R - l * \cos(\hat{a}) \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^c_3 T = \begin{bmatrix} \cos(\hat{\phi}_3) & -\sin(\hat{\phi}_3) & m * \sin(\hat{b}) \\ \sin(\hat{\phi}_3) & \cos(\hat{\phi}_3) & R - m * \cos(\hat{b}) \\ 0 & 0 & 1 \end{bmatrix}$$

Inverse Kinematics

$$\begin{bmatrix} x_1 \\ y_1 \\ \hat{\theta}_1 \\ x_2 \\ y_2 \\ \hat{\theta}_2 \\ x_3 \\ y_3 \\ \hat{\theta}_3 \end{bmatrix} = \begin{bmatrix} -R * \sin(\hat{\theta}_c) + x_c \\ R * \cos(\hat{\theta}_c) + y_c \\ \hat{\theta}_c + \hat{\phi}_1 \\ -l * \sin(a - \hat{\theta}_c) - R * \sin(\hat{\theta}_c) + x_c \\ -l * \cos(a - \hat{\theta}_c) + R * \cos(\hat{\theta}_c) + x_c \\ \hat{\theta}_c + \hat{\phi}_2 \\ m * \sin(b - \hat{\theta}_c) - R * \sin(\hat{\theta}_c) + x_c \\ -m * \sin(b - \hat{\theta}_c) + R * \cos(\hat{\theta}_c) + x_c \\ \hat{\theta}_c + \hat{\phi}_3 \end{bmatrix}$$

Example 9: Three Robot Cluster, Cluster Frame between R1 and R2, R3 follows Cluster Frame

Physical Diagram

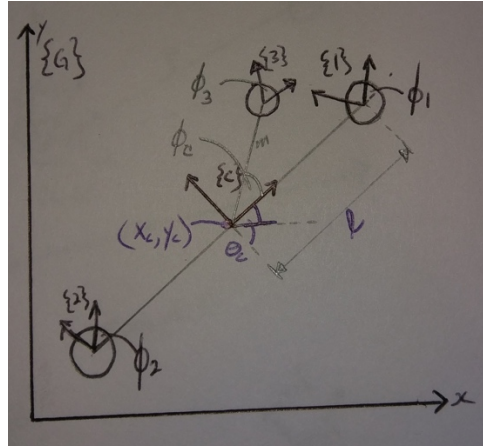


Figure 47: Physical Sketch of a three robot Cluster. Cluster frame is defined by frames 1 and 2, while the third frame follows that cluster frame.

Forward Kinematics

$$\begin{bmatrix} \hat{x}_c \\ \hat{y}_c \\ \hat{\phi}_1 \\ \hat{\phi}_2 \\ \hat{\theta}_c \\ \hat{l} \\ \hat{m} \\ \hat{\phi}_c \\ \hat{\phi}_3 \end{bmatrix} = \begin{bmatrix} \frac{(x_1 + x_2)}{2} \\ \frac{(y_1 + y_2)}{2} \\ \hat{\theta}_1 - \theta_c \\ \hat{\theta}_2 - \hat{\theta}_c \\ \text{atan2}((y_2 - y_1), (x_2 - x_1)) \\ \sqrt{(y_1 - y_c)^2 + (x_1 - x_c)^2} \\ \sqrt{(y_3 - y_c)^2 + (x_3 - x_c)^2} \\ \text{acos}\left(\frac{((y_3 - y_1)^2 + (x_3 - x_1)^2 - \hat{l}^2 - \hat{m}^2)}{2lm}\right) \\ \hat{\theta}_3 - \hat{\theta}_c \end{bmatrix}$$

Tree Structure

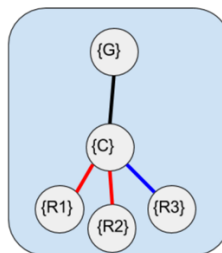


Figure 48: Cluster tree of a three robot Cluster. Cluster frame is defined by frames 1 and 2, while the third frame follows that cluster frame.

Transformation Matrices

$${}^c_c T = \begin{bmatrix} \cos(\hat{\theta}_c) & -\sin(\hat{\theta}_c) & \hat{x}_c \\ \sin(\hat{\theta}_c) & \cos(\hat{\theta}_c) & \hat{y}_c \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^c_1 T = \begin{bmatrix} \cos(\hat{\phi}_1) & -\sin(\hat{\phi}_1) & l \\ \sin(\hat{\phi}_1) & \cos(\hat{\phi}_1) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^c_2 T = \begin{bmatrix} \cos(\hat{\phi}_2) & -\sin(\hat{\phi}_2) & -l \\ \sin(\hat{\phi}_2) & \cos(\hat{\phi}_2) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^c_3 T = \begin{bmatrix} \cos(\hat{\phi}_3) & -\sin(\hat{\phi}_3) & m * \cos(\hat{\phi}_c) \\ \sin(\hat{\phi}_3) & \cos(\hat{\phi}_3) & m * \sin(\hat{\phi}_c) \\ 0 & 0 & 1 \end{bmatrix}$$

Inverse Kinematics

$$\begin{bmatrix} x_1 \\ y_1 \\ \hat{\theta}_1 \\ x_2 \\ y_2 \\ \hat{\theta}_2 \\ x_3 \\ y_3 \\ \hat{\theta}_3 \end{bmatrix} = \begin{bmatrix} l * \cos(\hat{\theta}_c) + x_c \\ l * \sin(\hat{\theta}_c) + y_c \\ \hat{\theta}_c + \hat{\phi}_1 \\ -l * \cos(\hat{\theta}_c) + x_c \\ -l * \sin(\hat{\theta}_c) + y_c \\ \hat{\theta}_c + \hat{\phi}_2 \\ m * \cos(\hat{\theta}_c + \hat{\phi}_c) + x_c \\ m * \sin(\hat{\theta}_c + \hat{\phi}_c) + y_c \\ \hat{\theta}_c + \hat{\phi}_3 \end{bmatrix}$$

Example 10: A Three Robot Cluster in a different definition

Physical Drawing:

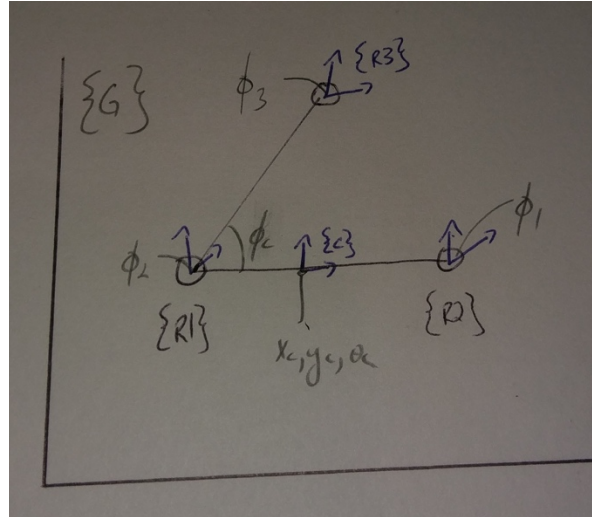


Figure 69: Physical View of a three robot Cluster. Cluster frame is defined by frames 1 and 2, while the third frame follows Robot 1

Forward Kinematics:

$$\begin{bmatrix} \hat{x}_c \\ \hat{y}_c \\ \hat{\phi}_1 \\ \hat{\phi}_2 \\ \hat{\theta}_c \\ \hat{l} \\ \hat{m} \\ \hat{\phi}_c \\ \hat{\phi}_3 \end{bmatrix} = \begin{bmatrix} \frac{(x_1 + x_2)}{2} \\ \frac{(y_1 + y_2)}{2} \\ \hat{\theta}_1 - \theta_c \\ \hat{\theta}_2 - \hat{\theta}_c \\ \text{atan2}((y_2 - y_1), (x_2 - x_1)) \\ \frac{1}{2} \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2} \\ \sqrt{(y_1 - y_3)^2 + (x_1 - x_3)^2} \\ \text{acos} \left(\frac{(y_3 - y_c)^2 + (x_3 - x_c)^2 - \hat{l}^2 - \hat{m}^2}{2lm} \right) \\ \hat{\theta}_3 - \hat{\theta}_c \end{bmatrix}$$

Graph Tree

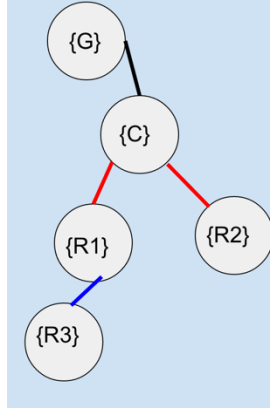


Figure 50: Cluster tree of a three robot Cluster. Cluster frame is defined by frames 1 and 2, while the third frame follows Robot 1

Transformation Matrices

$${}^G_C T = \begin{bmatrix} \cos(\hat{\theta}_c) & -\sin(\hat{\theta}_c) & x_c \\ \sin(\hat{\theta}_c) & \cos(\hat{\theta}_c) & y_c \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^C_{C1} T = \begin{bmatrix} 1 & 0 & l \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^{C1}_{R1} T = \begin{bmatrix} \cos(\hat{\phi}_1) & -\sin(\hat{\phi}_1) & 0 \\ \sin(\hat{\phi}_1) & \cos(\hat{\phi}_1) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^{C1}_{R2} T = \begin{bmatrix} \cos(\hat{\phi}_2) & -\sin(\hat{\phi}_2) & -l \\ \sin(\hat{\phi}_2) & \cos(\hat{\phi}_2) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^{C1}_{R3} T = \begin{bmatrix} \cos(\hat{\phi}_3) & -\sin(\hat{\phi}_3) & m * \cos(\hat{\phi}_c) \\ \sin(\hat{\phi}_3) & \cos(\hat{\phi}_3) & m * \sin(\hat{\phi}_c) \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^G_{R1} T = {}^G_C T * {}^C_{C1} T * {}^{C1}_{R1} T = \begin{bmatrix} \cos(\hat{\theta}_c + \hat{\phi}_1) & -\sin(\hat{\theta}_c + \hat{\phi}_1) & l * \cos(\hat{\theta}_c) + x_c \\ \sin(\hat{\theta}_c + \hat{\phi}_1) & \cos(\hat{\theta}_c + \hat{\phi}_1) & l * \sin(\hat{\theta}_c) + y_c \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^G_{R2} T = {}^G_C T * {}^{C1}_{R2} T = \begin{bmatrix} \cos(\hat{\theta}_c + \hat{\phi}_2) & -\sin(\hat{\theta}_c + \hat{\phi}_2) & -l * \cos(\hat{\theta}_c) + x_c \\ \sin(\hat{\theta}_c + \hat{\phi}_2) & \cos(\hat{\theta}_c + \hat{\phi}_2) & -l * \sin(\hat{\theta}_c) + y_c \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^G_{R3} T = {}^G_C T * {}^C_{C1} T * {}^{C1}_{R3} T = \begin{bmatrix} \cos(\hat{\theta}_c + \hat{\phi}_3) & -\sin(\hat{\theta}_c + \hat{\phi}_3) & m * \cos(\hat{\theta}_c + \hat{\phi}_c) + l * \cos(\hat{\theta}_c) + x_c \\ \sin(\hat{\theta}_c + \hat{\phi}_3) & \cos(\hat{\theta}_c + \hat{\phi}_3) & m * \sin(\hat{\theta}_c + \hat{\phi}_c) + l * \sin(\hat{\theta}_c) + y_c \\ 0 & 0 & 1 \end{bmatrix}$$

Inverse Kinematics

$$\begin{bmatrix} x_1 \\ y_1 \\ \hat{\theta}_1 \\ x_2 \\ y_2 \\ \hat{\theta}_2 \\ x_3 \\ y_3 \\ \hat{\theta}_3 \end{bmatrix} = \begin{bmatrix} l * \cos(\hat{\theta}_c) + x_c \\ l * \sin(\hat{\theta}_c) + y_c \\ \hat{\theta}_c + \hat{\phi}_1 \\ -l * \cos(\hat{\theta}_c) + x_c \\ -l * \sin(\hat{\theta}_c) + y_c \\ \hat{\theta}_c + \hat{\phi}_2 \\ m * \cos(\hat{\theta}_c + \hat{\phi}_c) + l * \cos(\hat{\theta}_c) + x_c \\ m * \sin(\hat{\theta}_c + \hat{\phi}_c) + l * \sin(\hat{\theta}_c) + y_c \\ \hat{\theta}_c + \hat{\phi}_3 \end{bmatrix}$$

Appendix B – Code used in Toolbox

B.1 Tree Data Structure Folder

Tree data Structure as MATLAB Class

By Jean-Yves Tinevez

13 Mar 2012 (Update 13 Feb 2016)

A per-value class that implements a generic tree data structure

<https://www.mathworks.com/matlabcentral/fileexchange/35623-tree-data-structure-as-a-matlab-class>

Description:

A tree is a hierarchical data structure where every node has exactly one parent (except the root) and no or several children.

Along with this relational structure, each node can store any kind of data.

This class implements it using plain MATLAB syntax and arrays. Most useful methods are implemented, using overloading of MATLAB functions for tree objects.

CJW Notes:

It is a rather intuitive class to use. For example, you can type `find((a.^2 .* b) > (c - 5) & d)`, with `a`, `b`, `c` and `d` being tree objects. Doing this will compute an actual solution for you. A rather long tutorial is included to walk you through these tree structures. It also shows several other features that were not utilized in this research. The tutorial can be found here: <http://tinevez.github.io/matlab-tree/>

B.2 Cluster_Builder Folder

B.2.1 cluster_tree.m

```
function [the_cluster, node_count, cluster_nodes, cc, cp ] = cluster_tree(  
the_cluster, clustercell, node_count, cluster_nodes, cc, cp )
```

```
    y = the_cluster.get(cp);  
    if cp>1  
        y=num2str(cell2mat(y));  
    end
```

```
    fprintf('How many cluster children does %s have \n', y )  
    prompt5 = input(' : ');
```

```
    for i=(cc):(prompt5+cc-1)  
        the_cluster = the_cluster.addnode(cp, clustercell(i));  
        node_count = node_count+1;  
        cluster_nodes = [cluster_nodes, node_count];  
    end
```

```
        cc= cc+prompt5;  
        cp = cp+1;
```

```
end
```

B.2.2 clusterbuilder2.m

```
function [ obj_out ] = clusterbuilder2( num_robots,num_clusters )

    % CLUSTERBUILDER2 - create your own cluster tree

    %num_robots = number of robots in the cluster
    %num_clusters = number of clusters to consider (main cluster +
subclusters)
    % Check number of inputs.

    if nargin > 2t
        error('clusterbuilder2 requires at most 2 inputs');
    elseif nargin < 2
        error('clusterbuilder 2 requires at least 3 inputs')
    end

    robotnamer           % Names the Robots R1, R2, R3, etc....
    clusternamer         % Names the Cluster Frames C, C1, C2, C3, etc...
    variablecreator      % Asks user for the cluster variables to be used
    symbolicbank        % Loads from this variable bank
    treecreator          % Arranges the tree into an Arborescence
    htmdefiner           % Arranges the HTM's
    msot                % Matlab struct with all information of the tree

end
```

B.2.3 clusternamer.m

```
% Creating a cell with the cluster frame names. Saves them symbollicaly.

for ii=1:(num_clusters+1)
    clusterarray(ii)= 'C';
end

clustercellinit = num2cell(clusterarray);
clustercell = genvarname(clustercellinit);
clustercell = clustercell(2:numel(clustercell));

for ii=1:(num_clusters)
    clustersyms(ii)= sym(clustercell(ii));
end
```


B.2.4 htmdefiner.m

```
% Defines the homogenous transforms for each edge in the arborescence

t = the_cluster;
namt=t;
fprintf('The Cluster tree we created is visible above. \n')
prompt6 = input('Enter the 3x3 or 4x4 T matrix from cluster frame to Ground
Frame \n >> ');
    tst = prompt6;
    sz = length(prompt6);
    if sz <3
        error('Not a Valid Homogeneous transform matrix');
    elseif sz > 4
        error('Not a Valid Homogeneous transform matrix');
    else
    end

t=t.set(1,prompt6);

for i=2:node_count
    var1 = t.get(i);
    var1=num2str(cell2mat(var1));
    var2 = t.getparent(i);
    if var2>1
        var2 = namt.get(var2);
        var2=num2str(cell2mat(var2));
    else
        var2 = namt.get(var2);
    end

    fprintf('What is the Transformation from %s to %s \n >>', var1, var2)
    subt = input(' : ');
    t=t.set(i,subt);
end
```

B.2.5 msot.m

```
% Creates a structure type data type containing all the tree information

obj_out.robotnodes=robot_nodes;
obj_out.clusternodes=cluster_nodes;
obj_out.htm_tree=t;
obj_out.txt_tree=the_cluster;
obj_out.var=variablesyms;
```

B.2.6 robot_tree.m

```
function [the_cluster, node_count, robot_nodes,rc] =
robot_tree(the_cluster,robotcell,node_count,robot_nodes, rc, cp)

    y = the_cluster.get(cp);
    if cp>1
        y=num2str(cell2mat(y));
    end

    fprintf('How many robot children does %s have? \n', y )
    prompt4 = input(' : ');

    for i=(1+rc):(prompt4+rc)
        the_cluster = the_cluster.addnode(cp, robotcell(i));
        node_count = node_count+1;
        robot_nodes = [robot_nodes, node_count];
        rc = rc+1;
    end

end
```

B.2.7 robotnamer.m

```
% This function names all the robots as R1, R2, R3 etc.....

for ii=1:(num_robots+1)
    robotarray(ii)= 'R';
end

robotcellinit = num2cell(robotarray);
robotcell = genvarname(robotcellinit);
robotcell = robotcell(2:numel(robotcell));

for ii=1:(num_robots)
    robotsyms(ii)= sym(robotcell(ii));
end

clear robotcellinit;
clear robotarray;
```

B.2.8 symbolicbank.m

```
% Lists some variables that can be used as cluster variables
% If your cluster variable is not listed, please add it here

syms a b c d e f g h i j k l m n o p q r s t u v w x y z;
syms A B C D E F G H I J K L M N O P Q R S T U V Q X Y Z;
syms alpha1 alpha2 alpha3 alpha4 alpha5 alpha6 alpha7 alpha8 alpha9;
syms beta1 beta2 beta3 beta4 beta5 beta6 beta7 beta8 beta9 beta10;
syms gamma1 gamma2 gamm3 gamma4 gamm5 gamma6 gamma7 gamma8 gamma9 gamma10;
syms theta1 theta2 theta3 theta4 theta5 theta6 theta7 theta8 theta9;
syms xc yc zc xc1 yc1 zc1 xc2 yc2 zc2 ;
syms phic phic1 phic2 phic3 phic4 phic5 phic6 phic7 phic8 phic9 phic10;
syms phic11 phic12 phic13 phic14 phic15 phic16 phic17 phic18 phic19 phic20;
syms phi1 phi2 phi3 phi4 phi5 phi6 phi7 phi8 phi9 phi10 phi11 phi12 phi13;
syms phi14 phi15 phi16 phi17 phi18 phi19 phi20 phi21 phi22 phi23 phi24;
syms tc tc1 tc2 tc3 tc4 tc5 tc6 tc7 tc8 tc9 tc10 tc11 tc12 tc13 tc14 tc15;
syms tc16 tc17 tc18 tc19 tc20;
syms aa bb cc dd ee ff gg hh ii jj kk ll mm nn oo pp qq rr ss tt uu vv;
syms ww xx yy zz;
syms AA BB CC DD EE FF GG HH II JJ KK LL MM NN OO PP QQ RR SS TT UU VV WW;
syms XX YY ZZ;
syms d1 d2 d3 d4 d5 d6 d7 d8 d9 d10 d11 d12 d13 d14 d15 d16
```

B.2.9 treecreator.m

```
%This step allows us to create a tree
% Let us create the base node, C, and start defining downward from there.

display('Let us now start creating our cluster tree')
the_cluster = tree('C');
node_count=1;
robot_nodes= [];
cc = 1; % Number of clusters we have assigned so far
rc = 0; %Robot counter
cp = 1 ;% Current Cluster position
cluster_nodes = 1;

while (numel(cluster_nodes)<num_clusters) || (numel(robot_nodes)<num_robots)

    [the_cluster, node_count, robot_nodes,rc] = ...
    robot_tree( the_cluster,robotcell,node_count,robot_nodes, rc,cp);

    [the_cluster, node_count, cluster_nodes, cc,cp ] = ...
    cluster_tree( the_cluster, clustercell,node_count,cluster_nodes, cc,cp);
end

fprintf(' \n \n \n \n ');
disp(the_cluster.tostring)
fprintf(' \n \n \n \n ');
```

B.2.10 variablecreator.m

```
% This program accepts the users input about which variables to use

display('What Cluster variables will you use?');
display('Please state your variables with no commas')
prompt3 = input('Example: a b c d xc yc phil \n >> ', 's');
prompt3 = strsplit(prompt3);

for ii=1:numel(prompt3)
    variablesyms(ii)= sym(prompt3(ii));
end

clear ii;
assume(variablesyms>0);           % Assumes variables are positive only
assumeAlso(variablesyms<0.1);    % Assumes variables are less than pi
```

B.3 The Inverse_Kinematics Folder

B.3.1 dirinvjac.m

```
function [ d_invjac, cluster ] = dirinvjac(cluster, opt1)

% This will use the htm to compute the inverse kinematics, then solve for
% the inverse Jacobian
% If 2 is chosen in the option 1 field, then the calculation assumed a 4x4
% transformation matrix that can be optimized for a 3x3 case.

    if nargin > 2
        error('dirinvjac requires at most 2 inputs');
    elseif nargin ==2
        [inv_kin, cluster] = invkin(cluster, opt1);
    elseif nargin ==1
        [inv_kin, cluster] = invkin(cluster);
    elseif nargin < 1
        error('dirinvjac requires at least 1 input')
    end

    d_invjac = jacobian(inv_kin,cluster.var);
    cluster.dirinvjac = d_invjac;
end
```

B.3.2 invkin.m

```
function [ inv_kin, cluster ] = invkin(cluster, opt1)

% This fxn uses the htm to get the inverse kinematics.
% It automatically defaults to option0 or option 1 based on the size of the
```

```

% homogeneous transform matrices
% Option 2 is an optimized version of Option 0. If selected, the program
% assumes the cluster is in the z=0 plane, and optimizes the matrix
% multiplication accordingly.

if nargin > 2
    error('invkin requires at most 2 inputs');

elseif nargin ==1
    tst = cluster.htm_tree.get(cluster.robotnodes(1));
    sz = length(tst);
    if sz == 4
        opt1 = 0;
    elseif sz == 3
        opt1= 1;
    else
        error('This cluster has invalid transformation matrices');
    end
elseif nargin < 1
    error('clusterbuilder 2 requires at least 1 input')
end

page = 1; y=5;          %Initializaing some variables

%Option1 == 0 - Case where the matrices are 4x4 matrices

if opt1==0

    for i=cluster.robotnodes
        y=i;
        x = cluster.htm_tree.get(i);
        y = cluster.htm_tree.getparent(i);
        while y>0
            x = cluster.htm_tree.get(y)*x;
            y = cluster.htm_tree.getparent(y);
        end

        T(:, :, page)=x;
        page = page+1;
    end

    inv_kin = [];
    page = page-1;

    for i=1:page
        ry = asin(-T(3,1,i));
        rz =asin(T(2,1,i)/cos(ry));
        rx =asin(T(3,2,i)/cos(ry));
        inv_kin = [inv_kin; T(1:3,4,i); rx;ry;rz];
        inv_kin = simplify(inv_kin);
    end
    cluster.inv_kin=inv_kin;

%Option1 ==1 - Case where the HTM Matrices are 3x3 case
elseif opt1==1

```

```

for i=cluster.robotnodes
    y=i;
    x = cluster.htm_tree.get(i);
    y = cluster.htm_tree.getparent(i);
    while y>0
        x = cluster.htm_tree.get(y)*x;
        y = cluster.htm_tree.getparent(y);
    end

    T(:, :, page)=x;
    page = page+1;
end

inv_kin = [];
page = page-1;

for i=1:page
    inv_kin = [inv_kin; T(1:2,3,i); acos(T(1,1,i))];
    inv_kin = simplify(inv_kin);
end
cluster.inv_kin=inv_kin;

%Option 2 - Solving the 4x4 case as the 3x3 case
else
    page =1;
    y=5;
    for i=cluster.robotnodes
        y=i;
        x = cluster.htm_tree.get(i);
        x = x([1,2,4],[1,2,4]);
        y = cluster.htm_tree.getparent(i);
        while y>0
            z= cluster.htm_tree.get(y);
            x = z([1,2,4],[1,2,4])*x;
            y = cluster.htm_tree.getparent(y);
        end
        T(:, :, page)=x;
        page = page+1;
    end

    inv_kin = [];
    page = page-1;

    for i=1:page
        inv_kin = [inv_kin; T(1:2,4,i); acos(T(1,1,i))];
        inv_kin = simplify(inv_kin);
    end
    cluster.inv_kin=inv_kin;
end

```

B.4 The Velocity_Propagation_Technique Folder

B.4.1 gettheRR3.m

```
function [ RR ] = gettheRR3( t, current_node)

% This function computes the product of all rotations from ground to
% current node.
% Calculates the product of rotations for the current node.
% Does Step A.2 of Section 3. Only works with 3 Degrees of Freedom (3DOF)

RR = eye(2);
y = t.getparent(current_node);

while y>0
    FTpar = t.get(y);
    Rpar = FTpar(1:2,1:2);
    RR = Rpar *RR;
    y= t.getparent(y);
end

RR = simplify(RR);

end
```

B.4.2 gettheRR6.m

```
function [ RR ] = gettheRR6( t, current_node)

% This function computes the product of all rotations from ground to
% current node.
% The same as gettheRR3, but is only called when dealing with more than
% 3DOF per robot.

RR = eye(3);
y = t.getparent(current_node);

while y>0
    FTpar = t.get(y);
    Rpar = FTpar(1:3,1:3);
    RR = Rpar *RR;
    y= t.getparent(y);
end

RR = simplify(RR);

end
```

B.4.3 getthev3.m

```
function [ ppjac ,px,py] = getthev3(current_node,t, i,variablesyms,dof)

% Calculates the local velocity of one frame relative to its parent.
% Performs step A.3

    trans = t.get(current_node);
    px = trans(1,3);
    py =trans(2,3);
    ppjac((1+(dof*(i-1))),,:) = jacobian(px,variablesyms);
    ppjac((2+(dof*(i-1))),,:) = jacobian(py,variablesyms);

end
```

B.4.4 getthev6.m

```
function [ ppjac ,px,py,pz] = getthev6( current_node,t, i,variablesyms,tpg)

% This function gets the local velocity of each robot. See also getthev3

    trans = t.get(current_node);
    px = trans(1,4);
    py =trans(2,4);
    pz = trans(3,4);
    ppjac((1+(tpg*(i-1))),,:) = jacobian(px,variablesyms);
    ppjac((2+(tpg*(i-1))),,:) = jacobian(py,variablesyms);
    ppjac((3+(tpg*(i-1))),,:) = jacobian(pz,variablesyms);

end
```

B.4.5 invjacfxn.m

```
function [ inv_jac_prop , cluster] = invjacfxn(cluster)
% This function solves for the inverse Jacobian using a propagation
% technique. It first checks the size of the T matrices, and executes the
% algorithm.
% This function, in its current form, does not have an optimization for 4x4
% matrices that are limited to the xy plane.
% Furthermore, this algorithm doesn't work if a cluster variable is used to
% define an actuation state.

% Determining to run 3 DOF case or 6 DOF
    sz = length(cluster.htm_tree.get(cluster.robotnodes(1)));
    if sz == 4
        opt1 = 0; dof = 6;
    elseif sz == 3
        opt1= 1; dof = 3;
    else
        error('This cluster has invalid transformation matrices');
    end
```



```

% Propagation Algorithm

if opt1 == 1

    for i=1:numel(cluster.robotnodes)
        current_node = cluster.robotnodes(i);
        parent_node = cluster.htm_tree.getparent(current_node);
        rrall = zeros(1,numel(cluster.var));
        rrall= sym(rrall);
        rrall2=rrall;

        while parent_node>0
            [RR] = gettheRR3( cluster.htm_tree,current_node);
% Product of rotations, RR
            [ppjac, px, py ] = getthev3( current_node,cluster.htm_tree,
i,cluster.var,dof); %local Velocity (linear component)
            v_rotated((1 + (dof*(i-1))),,:) = ppjac((1+(dof*(i-
1))),:)*(RR(1,1))+ ppjac((2+(dof*(i-1))),:)*RR(1,2); %Local Velocity
            v_rotated((2 + (dof*(i-1))),,:) = ppjac((1+(dof*(i-
1))),:)*(RR(2,1))+ ppjac((2+(dof*(i-1))),:)*RR(2,2); %Rotated
            [RRwxp, RRwyp] = rotatedwxp3( cluster.htm_tree, current_node,
cluster.var,RR,py,px ); % Global angular Velocity rotated
            rrall = rrall +(v_rotated((1+(dof*(i-1))),:) +RRwxp);
% Summing after each iteration
            rrall2 =rrall2 +(v_rotated((2+(dof*(i-1))),:) +RRwyp);
% Summing after each iteration
            current_node = parent_node;
% Updating current node
            parent_node = cluster.htm_tree.getparent(current_node);
%Updating parent node
        end

        [wvarray] = thefinv3( cluster.htm_tree, current_node ,cluster.var
); % Adding the final Velocity Term
        current_node = cluster.robotnodes(i);
% Recalling current node
        [wxarray] = thewthing3( cluster.htm_tree, current_node,
cluster.var); % Product of rotations

        inv_jac_prop((1+(dof*(i-1))),:) = rrall + wvarray(1,:); %
Assembling into one matrix
        inv_jac_prop((2+(dof*(i-1))),:) = rrall2 + wvarray(2,:); %
Assembling into one matrix
        inv_jac_prop((3+(dof*(i-1))),:) = wxarray; %
Assembling into one matrix

        cluster.propinvjac = inv_jac_prop; % Adds
jacobian to cluster structure

    end

else

```

```

for i=1:numel(cluster.robotnodes)
    current_node = cluster.robotnodes(i);
    parent_node = cluster.htm_tree.getparent(current_node);
    rrall = zeros(1,numel(cluster.var));
    rrall= sym(rrall);
    rrall2=rrall;
    rrall3 = rrall2;

    while parent_node>0
        [RR ] = gettheRR6( cluster.htm_tree,current_node);
        [ppjac, px, py,pz ] = getthev6(
current_node,cluster.htm_tree, i,cluster.var,dof);
        v_rotated((1 + (dof*(i-1))),:) = ppjac((1+(dof*(i-
1))),:)*(RR(1,1))+ ppjac((2+(dof*(i-1))),:)*RR(1,2) + ppjac((3+(dof*(i-
1))),:)*RR(1,3);
        v_rotated((2 + (dof*(i-1))),:) = ppjac((1+(dof*(i-
1))),:)*(RR(2,1))+ ppjac((2+(dof*(i-1))),:)*RR(2,2) + ppjac((3+(dof*(i-
1))),:)*RR(2,3);
        v_rotated((3 + (dof*(i-1))),:) = ppjac((1+(dof*(i-
1))),:)*(RR(3,1))+ ppjac((2+(dof*(i-1))),:)*RR(3,2) + ppjac((3+(dof*(i-
1))),:)*RR(3,3);
        [RRwxp, RRwyp, RRwzp ] = rotatedwxp6( cluster.htm_tree,
current_node, cluster.var,RR,py,px, pz );
        rrall = rrall +(v_rotated((1+(dof*(i-1))),:) +RRwxp);
        rrall2 = rrall2 +(v_rotated((2+(dof*(i-1))),:) +RRwyp);
        rrall3 = rrall3 + (v_rotated((3+(dof*(i-1))),:) +RRwzp);
        current_node = parent_node;
        parent_node = cluster.htm_tree.getparent(current_node);
    end

    [wvarray ] = thefinv6( cluster.htm_tree, current_node
,cluster.var );
    current_node = cluster.robotnodes(i);
    [wxarray ] = thewthing6( cluster.htm_tree, current_node,
cluster.var );

    inv_jac_prop((1+(dof*(i-1))),:) = rrall + wvarray(1,:);
    inv_jac_prop((2+(dof*(i-1))),:) = rrall2 + wvarray(2,:);
    inv_jac_prop((3+(dof*(i-1))),:) = rrall3 + wvarray(3,:);
    inv_jac_prop((4+(dof*(i-1))),:) = wxarray(1,:);
    inv_jac_prop((5+(dof*(i-1))),:) = wxarray(2,:);
    inv_jac_prop((6+(dof*(i-1))),:) = wxarray(3,:);
    cluster.propinvjac = inv_jac_prop;
end
end

end

% Further notes
% For each Robot in the cluster, the following Algorithm is compiled.
% V = V_final + sumall ((product of rotations)*(local_velocity +
(global_angular_velocity x local position) ) )

%However, the algorithm expands this as
%V = V_final + sumall ((product of rotations)*local_velocity + (product of
rotations)*(global_angular_velocity x local position) ) )

```

B.4.6 rotatedwxp3.m

```
function [ RRwxp, RRwyp ] = rotatedwxp3( t, current_node,
variablesyms,RR,py,px )

par =5;
wxarray = zeros(1,numel(variablesyms));

while par>0
    par = t.getparent(current_node);
    tpar = t.get(par);
    rz = acos(tpar(1,1));

    for j=1:numel(variablesyms)    % To find rz's position in cluster.var
        check = 2* (variablesyms(j)/rz);
        if check ==2
            break;
        else
            end
        end
    end

    wxarray(j) = 1;
    current_node = par;
    par = t.getparent(current_node);
end

RRwxp= wxarray* (-RR(1,1)*py +RR(1,2)*px);
RRwyp= wxarray* (-RR(2,1)*py +RR(2,2)*px);
end
```

B.4.7 rotatedwxp6.m

```
function [ RRwxp, RRwyp, RRwzp ] = rotatedwxp6( t, current_node,
variablesyms,RR,py,px, pz )
% This function gets the cross product, then multiplies it by the Rotation
% Matrices

par =5;
wxarray = zeros(3,numel(variablesyms));

while par>0
    par = t.getparent(current_node);
    tpar = t.get(par);
    ry = asin(-tpar(3,1));
    rz = asin(tpar(2,1)/cos(ry));
    rx = asin(tpar(3,2)/cos(ry));

    % Find Rx's position in the cluster.var
    for j=1:numel(variablesyms)
        check = 2* (variablesyms(j)/rx);
        if check ==2
            wxarray(1,j) = 1;
        else
            end
        end
    end
end
```

```

        wxarray(1,j) = 0;
    end
end

% Find Ry's position in the cluster.var
for k=1:numel(variablesyms)
    check = 2* (variablesyms(k)/ry);
    if check ==2
        wxarray(2,k) = 1;
    else
        wxarray(2,k) = 0;
    end
end

% Find Rz's position in the cluster.var
for l=1:numel(variablesyms)
    check = 2* (variablesyms(l)/rz);
    if check ==2
        wxarray(3,l) = 1;
    else
        wxarray(3,l) = 0;
    end
end

current_node = par;
par = t.getparent(current_node);
end

RRwxp= (RR(1,1)*(wxarray(2,:)*pz- wxarray(3,:)*py) +
RR(1,2)*(wxarray(3,:)*px- wxarray(1,:)*pz) + RR(1,3)*(wxarray(1,:)*py-
wxarray(2,:)*px));
RRwyp= (RR(2,1)*(wxarray(2,:)*pz- wxarray(3,:)*py) +
RR(2,2)*(wxarray(3,:)*px- wxarray(1,:)*pz) + RR(2,3)*(wxarray(1,:)*py-
wxarray(2,:)*px));
RRwzp= (RR(3,1)*(wxarray(2,:)*pz- wxarray(3,:)*py) +
RR(3,2)*(wxarray(3,:)*px- wxarray(1,:)*pz) + RR(3,3)*(wxarray(1,:)*py-
wxarray(2,:)*px));
end

```

B.4.8 thefinv3.m

```

function [ wvarray ] = thefinv3( t, current_node,variablesyms )
% This function calculates the final velocity

v1 = t.get(current_node); wxx = v1(1,3); wyy = v1(2,3);

% To find ww's position in variably syms
wvarray = zeros(1,numel(variablesyms));

for j=1:numel(variablesyms)
    check = 2* (variablesyms(j)/wxx);

```

```

        if check ==2
            break;
        else
            end
    end
end

for k=1:numel(variablesyms)
    check = 2* (variablesyms(k)/wyy);
    if check ==2
        break;
    else
        end
    end
end

wvarray(1,j) = 1;   wvarray(2,k) = 1;

end

```

B.4.9 thefinv6.m

```

function [ wvarray ] = thefinv6( t, current_node,variablesyms )
% This Function calculates the final velocity
    v1 = t.get(current_node);
    wxx = v1(1,4);   wyy = v1(2,4);   wzz = v1(3,4);

%To find ww's position in variably syms
    wvarray = zeros(1,numel(variablesyms));

    for j=1:numel(variablesyms)
        check = 2* (variablesyms(j)/wxx);
        if check ==2
            break;
        else
            end
        end
    end

    for k=1:numel(variablesyms)
        check = 2* (variablesyms(k)/wyy);
        if check ==2
            break;
        else
            end
        end
    end

    for l=1:numel(variablesyms)
        check = 2* (variablesyms(l)/wzz);
        if check ==2
            break;
        else
            end
        end
    end

    wvarray(1,j) = 1;   wvarray(2,k) = 1;   wvarray(3,l) = 1;

end

```

B.4.10 thewthing3.m

```
function [ wxarray ] = thewthing3( t, current_node, variablesyms )
%THEWTHING3 Calculates the angular velocity
% This program calculates the angular velocity for the cluster. It is
% called in the function invjacfxn.m

par=5;
wxarray = zeros(1,numel(variablesyms));
while par>=0
    tpar = t.get(current_node);
    ww = acos(tpar(1,1));

    %To find ww's position in variable syms
    for j=1:numel(variablesyms)
        check = 2* (variablesyms(j)/ww);
        if check ==2
            wxarray(j) = 1;
            break
        else
            % wxarray(j) = 0;
        end
    end
end

    if current_node==1
        break;
    else
        current_node = t.getparent(current_node);
        par = t.getparent(current_node);
    end
end
end
```

B.4.11 thewthing6.m

```
function [ wxarray ] = thewthing6( t, current_node, variablesyms )
%THEWTHING6 Calculates the angular velocity
% This function is called from the function invjacfxn.m

par=5;
wxarray = zeros(3,numel(variablesyms));
while par>=0
    tpar = t.get(current_node);
    ry = asin(-tpar(3,1));
    rz = asin(tpar(2,1)/cos(ry));
    rx = asin(tpar(3,2)/cos(ry));

    for j=1:numel(variablesyms) % Gets Ry's position in cluster.var
        check = 2* (variablesyms(j)/ry);
    end
end
```

```

        if check ==2
            wxarray(2,j) = 1;
            break;
        else
            % wxarray(2,j) = 0;
        end
    end

for k=1:numel(variablesyms) % Gets Rz's position in cluster.var
    check = 2* (variablesyms(k)/rz);
    if check ==2
        wxarray(3,k) = 1;
        break;
    else
        % wxarray(3,k) = 0;
    end
end

for l=1:numel(variablesyms) % Gets Rx's position in cluster.var
    check = 2* (variablesyms(l)/rx);
    if check ==2
        wxarray(1,l) = 1;
        break;
    else
        % wxarray(1,l) = 0;
    end
end

if current_node==1
    break
else
    current_node = t.getparent(current_node);
    par = t.getparent(current_node);
end
end
end
end

```

Appendix C - The Inverse Jacobian Formula: A Proof

Consider the HTM of a robot at hierarchical depth n , in a fixed frame, $\{i\}$, relative to global frame $\{G\}$. The HTM, G_iT , can be expanded into the form:

$${}^G_iT = {}^G_1T * (\prod_{j=2}^n {}^{C_{n-1}}_{C_n}T) * {}^{C_n}_iT \quad (\text{Eq 34})$$

This can be further expanded as follows:

$${}^G_iT = \begin{bmatrix} {}^G_1R & {}^G_1P \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} {}^1_2R & {}^1_2P \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} {}^2_3R & {}^2_3P \\ 0 & 1 \end{bmatrix} * \dots * \begin{bmatrix} {}^{i-1}_iR & {}^{i-1}_iP \\ 0 & 1 \end{bmatrix} \quad (\text{Eq 35})$$

$${}^G_iT = \begin{bmatrix} {}^G_1R^1_2R & {}^G_1R^1_2P + {}^G_1P \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} {}^2_3R & {}^2_3P \\ 0 & 1 \end{bmatrix} * \dots * \begin{bmatrix} {}^{i-1}_iR & {}^{i-1}_iP \\ 0 & 1 \end{bmatrix} \quad (\text{Eq 36})$$

$${}^G_iT = \begin{bmatrix} {}^G_1R^1_2R^2_3R \dots {}^{i-1}_iR & {}^G_1R^1_2R^2_3R \dots {}^{i-2}_{i-1}R {}^{i-1}_iP + \dots + {}^G_1R^1_2P + {}^G_1P \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} {}^G_iR & {}^G_iP \\ 0 & 1 \end{bmatrix} \quad (\text{Eq 37})$$

By taking a look at the final matrix, the position vector G_iP describing the position of frame $\{i\}$ relative to $\{G\}$ can be recovered:

$${}^G_iP = {}^G_1R^1_2R^2_3R \dots {}^{i-2}_{i-1}R {}^{i-1}_iP + \dots + {}^G_1R^1_2P + {}^G_1P \quad (\text{Eq 38})$$

This can be written more compactly as:

$${}^G_nP = \sum_{i=1}^{n-1} (\prod_j^{i-1} {}^j_{j+1}R) {}^{i+1}_iP + {}^G_1P \quad (\text{Eq 39})$$

Taking a derivative of the above expression, the following is obtained:

$${}^G_i\dot{P} = \sum_{i=1}^{n-1} (\prod_j^{i-1} {}^j_{j+1}R) ({}^{i+1}_i\dot{P} + ({}^G_i\omega \times {}^{i-1}_iP)) + {}^G_1\dot{P} \quad (\text{Eq 40})$$

This equation is identical to the equation that was obtained via the propagation technique (Equation 23).

Let us now consider the final matrix also provides the rotation matrix, G_nR , which gives the rotation of robot n relative to the global frame.

$${}^G_nR = {}^G_1R^1_2R^2_3R \dots {}^{i-1}_nR \quad (\text{Eq 41})$$

From this a new vector can be created to represent these rotations:

$${}^G_n\theta = \begin{bmatrix} \alpha_z \\ \beta_y \\ \gamma_x \end{bmatrix} \quad (\text{Eq 42})$$

Where $\alpha_z, \beta_y, \gamma_x$ are the sum of all rotations about the z , y , and x axis respectively. For example, all the rotations about the z axis are found by:

$${}^G_nR_z = \begin{bmatrix} {}^G_1R_z({}^G_1\theta) \\ 0 \\ 0 \end{bmatrix} + {}^G_1R * \begin{bmatrix} {}^1_2R_z({}^1_2\theta) \\ 0 \\ 0 \end{bmatrix} + \dots + {}^G_1R * {}^1_2R * \dots * {}^{n-2}_{n-1}R * \begin{bmatrix} {}^{n-1}_nR_z({}^{i-1}_i\theta) \\ 0 \\ 0 \end{bmatrix} \quad (\text{Eq 43})$$

This can be written in compact form as

$${}^G_nR_z = \sum_{n=1}^k (\prod_{i=1}^n {}^{i-2}_{i-1}R) ({}^{i-1}_i\theta) \quad (\text{Eq 44})$$

Taking a derivative of this yields:

$${}^G_n\dot{R}_z = {}^G_k\vec{\omega} = \sum_{n=1}^k (\prod_{i=1}^n {}^{i-2}_{i-1}R) ({}^{i-1}_i\dot{\theta}) \quad (\text{Eq 45})$$

This equation is also identical to the equation that was obtained via the propagation technique (Equation 24).

Arriving at the propagation equations by taking derivatives of the inverse kinematics shows that the propagation method of computing the inverse kinematic equations is valid.