3-13-2011

Multirate Control Applied to a Cluster Space Robot Formation

Michael Charles Meserve
Santa Clara University

Follow this and additional works at: https://scholarcommons.scu.edu/elec_mstr

Recommended Citation
https://scholarcommons.scu.edu/elec_mstr/4
SANTA CLARA UNIVERSITY
Department of Electrical Engineering

Date: March 13th, 2011

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Michael Charles Meserve

ENTITLED

Multirate Control Applied to a Cluster Space Robot Formation

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

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

Christopher Kitts, Thesis Advisor

Cary Y. Yang, Department Chair
Multirate Control Applied to a Cluster Space Robot Formation

By

Michael Charles Meserve

GRADUATE MASTER THESIS

Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Electrical Engineering in the School of Engineering Santa Clara University, 2011

Santa Clara, California
Abstract

Cluster space control allows for the precise guidance of a formation of robots. The desired formation geometry is achieved by fixing the positions of individual robots relative to a mobile cluster space coordinate system. The formation is controlled by moving the cluster space origin along a trajectory defined in global coordinates. Robot positions are translated between cluster space and global coordinates by a set of nonlinear kinematic equations; robot velocities are translated by the Jacobian matrix derived from the kinematic equations. Calculation of the Jacobian matrix becomes computationally costly as the number of robots and degrees of freedom increase, which may limit the number of robots in a formation or the accuracy with which they can be controlled.

Multirate control is a method of reducing the computation required for a control loop. The key concept of multirate control is that some portions of a control loop may not need to run as fast as other portions. Partitioning the loop so that some portions can run at a slower rate reduces the overall computational load. The ratio of the fast loop rate to the slow rate is the parameter $m$; higher values of $m$ mean reduced computation of the slow parts of the loop. In cluster space control, the Jacobian matrix often changes very little between successive iterations of the control loop, suggesting that this expensive calculation can be performed at a slower rate. This thesis explores the potential for multirate control to reduce the computational load of a cluster space control system.
A three-robot cluster space simulation with nine degrees of freedom has been modified to implement multirate control. The ratio $m$ of the main control loop update rate to the update rate of the 9-by-9 Jacobian matrix can be varied to see the effect of multirate control on the error, defined as the difference between the desired and actual trajectories. Formation velocity, shape, and $m$ were parameterized and a test program created to repeatedly run the simulation while varying $m$ and any other single parameter. To graphically display the results, $m$ and the variable parameter are used as the horizontal axes of a surface plot whose height is the peak error during the simulation run.

It was found that proximity to a singularity in the Jacobian requires a faster update rate, diminishing the savings allowed by multirate control. An alternate cluster space orientation which eliminates the singularity allows for a slower update rate, with increased computational savings.

It is shown that the expected computational savings from multirate control is highest in systems for which the Jacobians are dense. As Jacobian sparsity increases, the computational savings offered by multirate control is reduced.
Acknowledgements

This thesis would not have been possible without the patience and encouragement of Professor Christopher Kitts. Also, it relies heavily on previous work by and assistance from Ignacio Mas of the Robotic Systems Laboratory at Santa Clara University.
# Table of Contents

Abstract.................................................................................................................. iii
Acknowledgements................................................................................................... vi
Table of Contents ...................................................................................................... vi
List of Equations ........................................................................................................ vii
List of Figures ............................................................................................................ vii
List of Tables ............................................................................................................. vii

1 Introduction ............................................................................................................. 1
   1.1 Formation Control ............................................................................................ 1
   1.2 Cluster Space Control ...................................................................................... 2
   1.3 Computational Issues ...................................................................................... 3
   1.4 Multirate Control ............................................................................................ 4
   1.5 Problem Statement ......................................................................................... 5

2 Cluster Space Control .............................................................................................. 6
   2.1 Theory of Cluster Space Control ..................................................................... 6
   2.2 Three-Robot Cluster Space Implementation .................................................. 7
   2.3 Multirate Systems .......................................................................................... 8

3 Simulation Description ............................................................................................ 10
   3.1 Simulink Model .............................................................................................. 10
   3.2 Example Simulation Runs .............................................................................. 13
      3.2.1 Example 1 - Cluster Velocity and Ratio $m$ ............................................ 13
      3.2.2 Example 2 - Angle $\beta$ ....................................................................... 19
      3.2.3 Example 3 - Position Measurement Error as Noise in $R$ ....................... 21
      3.2.4 Example 4 - Shape Mode for Reduced Condition Number ..................... 22
      3.2.5 Example 5 - Rate of Change of $\beta$: $\omega$ ......................................... 23
   3.3 Estimate of Computational Savings ................................................................. 25

4 Summary .................................................................................................................. 29
   4.1 Future work .................................................................................................... 29
      4.1.1 Hardware Implementation ....................................................................... 30
      4.1.2 Different Choices of Cluster Shape Variables ......................................... 30
      4.1.3 Transparent Shape Definition Change .................................................... 31

5 References .............................................................................................................. 32

Appendix: Software Description .............................................................................. 33
File Listing ................................................................................................................. 33
   New Files ............................................................................................................ 33
   Changed Files .................................................................................................... 34
   Deleted Files ..................................................................................................... 35
   Unchanged Files ................................................................................................ 35
   Instructions .......................................................................................................... 35
List of Equations

Forward Kinematic Equation (2-1) ......................................................... 7
Inverse Kinematic Equation (2-2) .......................................................... 7
Time Derivative Forward Kinematic Equation (2-3) ............................... 7
Time Derivative Inverse Kinematic Equation (2-4) .................................. 7
W, Cluster-Space Mass Matrix (3-1) ....................................................... 27
F<sub>W</sub>, W- Contribution to Force Vector (3-2) ....................................... 27
α, Velocity-Dependent Effects Matrix (3-3) ............................................ 27
F<sub>α</sub>, α- Contribution to Force Vector (3-4) ........................................... 27

List of Figures

Figure 1.4-1: A simplified cluster space control loop................................4
Figure 2.2-1: Three-robot cluster space implementation.......................... 8
Figure 3.1-1: Cluster space implementation ........................................... 11
Figure 3.1-2: Virtual Reality display of Simulink model.......................... 12
Figure 3.1-3: Alternate orientation of the cluster .................................... 12
Figure 3.2-1: Example 1 (velocity=1, m=1) ........................................... 14
Figure 3.2-2: Cluster velocity=1, m=1 .................................................... 14
Figure 3.2-3: Jacobian condition numbers ............................................. 15
Figure 3.2-4: Cluster velocity=2, m=20 ............................................... 15
Figure 3.2-5: Cluster velocity=3, m=60 ............................................... 16
Figure 3.2-6: Jacobian condition numbers ............................................. 16
Figure 3.2-7: Surface Plots ................................................................. 18
Figure 3.2-8: Surface plot of the maximum condition number of the Jacobian 19
Figures 3.2-9: Results of a single run with β=176° and cluster velocity equal to one... 20
Figure 3.2-10: Errors in p and q at β=178°, 179° and 179.8°, respectively .. 20
Figure 3.2-11: Maximum error and Jacobian condition number ............... 21
Figure 3.2-12: Example 3: Noise enabled............................................. 21
Figure 3.2-13: Example 4: β = 180 degrees, using ShapeMode=2 (noise enabled) ... 22
Figure 3.2-14: Example 4: Maximum error and Jacobian condition number ... 23
Figure 3.2-15: Example 5, VRML display ........................................... 24
Figure 3.2-16: Example 5, m = 1 ......................................................... 24
Figure 3.2-17: Example 5, m = 20 ....................................................... 25
Figure 3.3-1: Three-robot cluster space implementation (repeated) .............. 26
Figure 4.1-1: Alternate choices of cluster shape variables .......................... 30

List of Tables

Table 3.3-1: Operations for a Single Control Loop Iteration ....................... 28
1 Introduction
Having gained the ability to control a single robot, a logical next step would be to try controlling a formation of multiple robots working in concert. This paper is interested in the challenges of controlling a formation of mobile robots. In particular, it applies multirate control, in which computationally expensive parts of a control loop run at a slower rate, to a formation control technique known as cluster space control.

1.1 Formation Control
There are many techniques for controlling a formation of robots; the choice of technique will depend on both the abilities of the individual robots as well as the task to which they are being applied. Each robot may know the position and velocity of every other robot in the system, or only a select few robots, or no other robots. The communications between robots may have high bandwidth, low latency, and be bidirectional, or may be slow, delayed, and one-way. The robots may have powerful computing systems that can handle complex algorithms, or (due to weight and power constraints) have minimal processing power. The trajectory to be followed may be precisely known in advance, or it may be determined by conditions encountered during the mission. These and other variables determine the best technique for a given application.

One technique, mimicking the natural behavior seen in flocks of birds and other animals, is called "Leader-Follower" [9]. The trajectory is known only to a single robot (the "leader"); the remaining robots ("followers") maintain their position relative to neighboring robots and obstacles. Such a system of robots can display complex behavior, such as dividing to go around an obstacle and merging back together, without centralized control of individual robots. However, if the lead robot were to fail, the entire system would be halted.

A similar technique [10] avoids this single point of failure by not having any lead robot; instead, the perception of a "virtual leader" is sent to every robot in the formation. All robots behave as followers and treat the virtual leader as another robotic neighbor. The
real robots can be controlled by moving the virtual leader (which actually means changing the position and velocity the followers perceive the virtual leader as having) along the desired trajectory.

These techniques have the advantage of minimizing the computational load of centrally controlling large numbers of individual robots. However, each robot must be able to sense the distance and direction to its neighbors. Also, there is no assurance that an exact trajectory will be followed by any individual robot. If such precision is required, it would be difficult or impossible for a human operator to individually direct multiple robots. *Cluster space control* [11] is a formation control technique which allows for such precision, while making control simple for a human operator.

### 1.2 Cluster Space Control

An intuitive way to completely describe a cluster of robots is simply a vector of each robot’s position and orientation relative to some global coordinate system. This is called the global pose of the cluster, or sometimes the *robot space* pose, as the robots operate in global coordinates.

In cluster space control, the system of robots is arranged in their desired positions and orientations relative to a set of mobile cluster-specific axes. The operator, which may be either a human being or an automated agent, controls the position and orientation of these axes, relative to the global coordinate system. The vector combining the global position and orientation of the cluster-specific axes and the shape variables (which describe the robot(s) positions and orientations relative to those axes) is the *cluster space* pose of the system. Just as the time derivative of the global pose is the global velocity, the time derivative of the cluster space pose is the cluster space velocity. Together, the cluster space pose and velocity completely describe the state of the system, as do the global pose and velocity, and either method may be used as the basis of a formation control system.
From the perspective of the operator, the cluster is moved as a single entity, by specifying the desired position and orientation of the cluster space axes. The shape of the cluster depends only on the shape variables, and can be changed independent of the motion of the cluster origin. For instance, the cluster can be rotated about or translated relative to the cluster origin, or scaled to cover a larger or smaller area. Individual robots can be rotated on their own axes, or moved relative to the cluster space axes. The desired positions and orientations form the trajectory for the cluster to follow.

The cluster space feedback controller operates on the difference between the desired and the actual cluster space states of the robots, generating a control signal in cluster space coordinates. This control signal must be converted to global coordinates to be understood by the robots.

1.3 Computational Issues

In addition to the calculations of a nonlinear feedback control loop, cluster space control adds the task of converting the robots’ states between global and cluster coordinates. This conversion requires the calculation of a Jacobian matrix, which can be computationally expensive. Because the size of the Jacobian matrix increases with the square of the number of robots in the system, this computational expense may be a limiting factor for controller performance.

If the controller is centralized, it will not have the same weight and power limitations that a controller distributed among the robots will have. A tremendous amount of computer power may be obtained for relatively little cost. Still, for field applications, a central controller may need to be mobile, and this will impose practical limits of weight and battery life.

A decentralized controller may distribute the conversion between global and cluster space among the robots in the formation; proper choice of shape variables may allow each robot to be responsible for converting only a limited subset of the state. However, computing
resources on a single robot will be more limited than a central processor, and given enough robots and degrees of freedom, we may still encounter a limit where the control loop cannot operate fast enough to achieve the desired accuracy.

Efficient code implementation can go a long way to reducing the number of calculations necessary to compute the Jacobian. Many terms in the equation of a Jacobian matrix appear multiple times, and careful reuse of intermediate results can reduce the number of expensive function calls. Careful inspection of the control loop reveals another potential code optimization, known as multirate control.

### 1.4 Multirate Control

A simple representation of the cluster space control loop is shown in Figure 1.4-1. The Jacobian matrix is used in the conversion of the robot states, which are sensed in the global coordinate system, to the cluster space coordinate system (in which the trajectory is defined and the control loop operates). The resulting control signals are converted—again using the Jacobian—back to global coordinates which the robots understand.

![Figure 1.4-1: A conceptual cluster space control loop.](image)

The control loop must operate fast enough to keep the robots on the desired trajectory. Calculating the Jacobian on every loop iteration imposes a large computational burden, limiting the speed of the control loop.

However, the Jacobian is a function of the robot positions, which may not change very rapidly. If the calculation of the Jacobian can be skipped for some number of loop
iterations, a large reduction in computation will result. Running different parts of a control loop at different rates is known as multirate control.

1.5 Problem Statement

To investigate the degree to which multirate control can reduce the computational load required for cluster space control, a simulation architecture which allows for multirate operation has been developed and used to perform an initial performance study. To create this, an existing Simulink model of a three-robot cluster-space system [8] was modified to allow multirate operation, where the ratio of the fast loop rate to the Jacobian update rate is $m$. At each step in the simulation where a Jacobian or inverse Jacobian is calculated, the result is saved and reused for $m-1$ cycles. When $m$ is equal to one, the simulation behaves normally; as $m$ is increased, the effect of multirate control is observed.

In addition to the multirate behavior, a number of features were added to facilitate research using this simulation. Plots can be made which analyze a single run, or which combine a large number of single runs to show trends as $m$ and other parameters are varied. Noise can be added to the position measurements to simulate real-life sensor inaccuracy. The geometric definition of the robot cluster can be altered to avoid a singularity.

The simulations suggest that a high value of $m$ can be used in some circumstances, greatly reducing computational load. The value of $m$ must be reduced as cluster velocity increases. Further, while $m$ is limited in the vicinity of a singularity, an alternate definition of the cluster geometry which eliminates the singularity allows for large $m$ values.

The number of computations per control loop iteration can be reduced by a factor approaching $m$ for systems with large numbers of robots and dense Jacobian matrices.
2 Cluster Space Control

To demonstrate how multirate control is applied to a cluster space controller, we first explore the theory of cluster space control, then the specific implementation (a simulated formation of three robots confined to a plane, each with three degrees of freedom) used in this paper. Finally, we describe some other multirate implementations and their choices for the value of $m$.

2.1 Theory of Cluster Space Control

Given a number of robots that must cooperate to complete a task, it is convenient for an operator to work with the group of robots as a single entity, and let an automated system translate control of the group into commands for each individual robot. One such approach is cluster space control [11][7]. We define a group of robots as a cluster, with the position and orientation of each robot expressed in the form of a cluster geometry and individual robot orientations with respect to the cluster frame. The origin of the cluster frame has a position and orientation ($<X_c, Y_c, \theta_c>$, in a 2-dimensional system) relative to some global coordinate system; additional cluster space variables describe the pose, forming a vector $C$ which completely describes the cluster. The operator controls the motion and pose of the cluster by setting a trajectory of desired values of $C$ and its derivative, $\dot{C}$.

Each robot can also be described in terms of its position and orientation ($<X_n, Y_n, \theta_n>$, in two dimensions, where $n$ ranges from one to $N$ (the number of robots in the system)) relative to the global coordinate system; these coordinates form the vector $R$, which also completely describes the cluster position. $R$ and $C$ are the same length, and are related by a set of $N$ forward kinematic equations expressing each element of $C$ as a function of $R$, and $N$ inverse kinematic equations expressing each element of $R$ as a function of $C$; these are equations 2-1 and 2-2 respectively, below.
As each robot will be in motion, there is also a vector \( \hat{\mathbf{R}} \) describing the velocity of the system in global coordinates. Similarly, there is a vector \( \hat{\mathbf{C}} \) giving the cluster space velocity. These vectors are related by the Jacobian matrix \( \mathbf{J} \) and its inverse \( \mathbf{J}^{-1} \), derived from the kinematic equations relating \( \mathbf{R} \) and \( \mathbf{C} \); equations 2-3 and 2-4, below, show these relationships. Both systems of expressing the robot system are needed: The human operator interacts with the cluster description \( \mathbf{C} \) and \( \hat{\mathbf{C}} \), but the individual robots must operate in terms of \( \mathbf{R} \) and \( \hat{\mathbf{R}} \) to deal with true forces and velocities in the global coordinate system.

\[
\begin{align*}
\hat{\mathbf{C}} &= f(\hat{\mathbf{R}}) \quad (2-1) \\
\hat{\mathbf{C}} &= \mathbf{J}\hat{\mathbf{R}} \quad (2-3) \\
\mathbf{R} &= f^{-1}(\mathbf{C}) \quad (2-2) \\
\hat{\mathbf{R}} &= \mathbf{J}^{-1}\hat{\mathbf{C}} \quad (2-4)
\end{align*}
\]

### 2.2 Three-Robot Cluster Space Implementation

The control scheme shown in Figure 2.2-1 below depicts a partitioned, nonlinear dynamic cluster-space controller for a three-robot cluster confined to motion on a plane [8][13]. This is not the only possible controller implementation for this robot cluster. A non-partitioned linear kinematic controller is described in [2], and a partitioned nonlinear kinematic controller is also developed in [13]. The implementation in Figure 2.2-1 was chosen due to the availability of a high-quality simulation to which to a multirate feature could be added.

On the left side of Figure 2.2-1, the trajectory (subscript des for desired) is provide by the user in cluster-space coordinates. The PID controller accepts both the desired trajectory and the actual position and velocity in cluster-space coordinates. The controller output \( \mathbf{F}_{\text{PID}} \) is multiplied by the \( \mathbf{W} \) matrix (the cluster space equivalent of mass matrix \( \mathbf{M} \)), resulting in \( \mathbf{F}_W \). Also, the \( \mathbf{a} \) matrix (which offsets the velocity-dependent effects of \( \mathbf{M} \) and friction matrix \( \mathbf{B} \)) is calculated to produce \( \mathbf{F}_a \) and added to \( \mathbf{F}_W \). The resulting cluster space force vector \( \mathbf{F}_C \) is converted to a global coordinate “torque” vector \( \mathbf{\tau} \) by \( \mathbf{J}^T \), and each robot responds just to its own commanded global torque (which will tend to reduce the cluster-space error). The updated position and velocity in global coordinates (\( \mathbf{R} \) and
\( \dot{\mathbf{R}} \), on the far right) are then converted back to cluster space via the forward kinematic equations and \( \mathbf{J} \), completing the loop.

Each loop iteration requires calculating \( \mathbf{J} \), \( \mathbf{J}^{-1} \), and \( \mathbf{J}^T \), for coordinate translations (from \( \dot{\mathbf{R}} \) to \( \dot{\mathbf{C}} \) and from \( \mathbf{F}_C \) to \( \tau \)) and for calculating \( \mathbf{W} \) and \( \alpha \). As the Jacobian is a function only of the cluster pose, it changes slowly relative to the speed of the control loop. This suggests that the Jacobian need not be recalculated every loop iteration, a concept known as multirate control. Further, \( \mathbf{W} \) and \( \alpha \) are dependent only on the Jacobians and constants, suggesting that these expensive calculations may take advantage of multirate control as well.

![Diagram](image)

*Figure 2.2.1: Three-robot cluster space implementation. Solid lines are the fast control loop (10 Hz); dotted lines are calculations dependent on \( \dot{\mathbf{R}} \) which may be done at a slower rate. Based on Figure 7 from [13] and Figure 10.15 from [6].*

### 2.3 Multirate Systems

The idea of calculating one part of a control loop at a slower rate than the main loop is called multirate control. The ratio of fast frequency to slow frequency is designated \( m \), which must be an integer for a discrete system. This thesis explores the application of multirate control to a particular cluster space control system, but the concept predates cluster space control and has been successfully applied to other control schemes.
Examples of multirate manipulators abound. [1] describes a PUMA 560 robot with a 100 Hz slow loop and 200 Hz fast loop. [6] suggests a fast loop at 500 Hz or higher with a slow loop at 100 Hz. In [3], multirate control is achieved via a decimator and interpolator at the input and output, respectively, of a slow component in parallel with a fast component; the interpolator simply injects the last calculated value repeatedly at the fast loop rate until a new value is calculated. The application is a hard drive head positioning controller with fast rate of 50 kHz and one-third of that for the slow rate; however, no methodology is given for how either the fast rate or the ratio was arrived at.

[4] suggests selecting a fast rate of 10 times the cutoff frequency of the closed-loop system, and gives a procedure for choosing \( m \) (capital M is used in [4]). The system is partitioned so that the fastest mode (or modes, if the highest frequencies are close together) are separated from the remaining slow modes. The ratio is then taken as the number of unique eigenvalues of the slow partition. This allows each mode in the slow partition to be calculated during one fast time step within the slow cycle, spreading the computation out evenly (this is referred to as interlacing). The given example uses a 100 Hz fast rate for an aircraft controller, and lets \( m=11 \) to match the number of unique eigenvalues of the slow modes.

For the three-robot system used in this paper, we allow the ratio \( m \) to vary so we can determine the effect of \( m \) on the ability of the cluster space control system to track a trajectory. This will provide a foundation for future work applying the multirate concept to cluster space control.
3 Simulation Description

3.1 Simulink Model

The three-robot Simulink model described in [8] was obtained from the author, modeling the control scheme shown in Figure 2.2-1. This Simulink model was modified to only update the Jacobian, inverse Jacobian, and transpose Jacobian every \( m \) iterations, where \( m \) can be specified by the user.

The simulation runs at a fixed step size of 0.1 seconds (10 Hz). To implement multirate control, every \( m^{th} \) Jacobian and inverse are stored and reused for the next \( m-1 \) cycles; this reduces the update rate for just the Jacobian recalculations, while the control loop continues at the faster rate. When multiplying a velocity vector by \( J \) or \( J^{-1} \), the vector itself is always current; it is only the Jacobian that may be reused.

The three robots in the cluster are arranged as shown in Figure 3.1-1a. In global coordinates (or “robot-space” coordinates), the cluster pose is defined by a nine-element vector \( \mathbf{R} \), consisting of the XY global position and orientation of each robot: \( < X_1, Y_1, \theta_1, X_2, Y_2, \theta_2, X_3, Y_3, \theta_3 > \). In the particular cluster space configuration used in this thesis, the origin of the cluster-space coordinate system is at \{C\}, the centroid of the triangle formed by the three robots. The cluster Y-axis is the line from the centroid through Robot 1. The positions of Robots 2 and 3 are defined by the distances \( p \) and \( q \), and the angle \( \beta \). Each angle \( \varphi_n \) describes the orientation of Robot \( n \) relative to the cluster Y-axis. The complete cluster pose can thus be defined by a nine-element vector \( \mathbf{C}: < X_C, Y_C, \theta_C, \varphi_1, \varphi_2, \varphi_3, p, q, \beta > \). There are many other possible ways to express \( \mathbf{R} \) in terms of a cluster space \( \mathbf{C} \) vector [9]; this configuration is used exclusively in this thesis and accompanying simulation.

In all trajectories used in this thesis, the three-robot cluster forms an isosceles triangle, moving along its line of symmetry. The default orientation (3.1-1a) sets Robot 1 in the
lead position with the cluster Y-axis as the line of symmetry, with \( p \) and \( q \) equal. \( \theta_C \) and all \( \phi_n \) are all set to zero, so that all three robots point in the direction of forward motion. This orientation is chosen in the simulation by setting the \textit{ShapeMode} flag to 1.

An alternative orientation, shown in Figure 3.1-1b, can be selected by setting \textit{ShapeMode} to 2. In this trajectory, the cluster is rotated by \( \theta_C \) so that Robot 2 becomes the leading robot. The cluster Y-axis is no longer the direction of forward motion; the cluster moves along the line from the centroid to the center of Robot 2. Distances \( p \) and \( q \) are no longer equal, and the angles \( \phi_n \) are all set to \(-\theta_C\), which keeps the robots pointing in the direction of forward motion. Note that the vector \( C \) still consists of the same nine variables and the kinematic equations relating \( R \) and \( C \) are unchanged; the only change is in the trajectory and the roles played by Robot 1 and Robot 2. This alternate orientation is explored further in Example 4 (see section 3.2.4).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure3_1.png}
\caption{Cluster space implementation. The centroid of the triangle formed by the three robots is the origin of the cluster space coordinate axes. In the default configuration (\textit{ShapeMode} = 1, \textit{a}), the cluster moves “forward” in the direction of Robot 1. Alternatively, the cluster may be rotated (\textit{ShapeMode} = 2, \textit{b}) so that Robot 2 is the lead robot. Based on Figure 4 from [2].}
\end{figure}

A trajectory generator guides the centroid of the triangle formed by the three robots in the cluster along a straight line, through a 180\(^\circ\) turn, and again along a straight line in the opposite direction (See Figure 3.1-2, (a) through (c) below). In the default orientation, Robot 1 is in the lead position.
Figure 3.1-2: Virtual Reality display of Simulink model. Successive frames show the cluster proceeding in a straight line at the start of the trajectory (a), halfway through the 180-degree turn (b), and after completing the turn (c). Robot 1 (red) is in the lead. Note the small arrow at the centroid, which always points in the direction the cluster is moving, along the line from the centroid to Robot 1.

Variables are the cluster linear velocity, the angle $\beta$, angular velocity $\omega$ (the rate of change of $\beta$), and the ratio $m$. Also, the orientation of the cluster relative to the trajectory may be rotated to place Robot 2 in the lead position, as shown in Figure 3.1-3.

Figure 3.1-3: Alternate orientation of the cluster. Robot 2 (green) is now in the lead, and Robot 1 (red) is now on the inside of the turn where Robot 3 (blue) was previously. Note the small arrow at the centroid continues to point towards Robot 1 (along the cluster coordinate Y-axis), not in the direction the cluster is moving.
From a single run of the model, a detailed set of plots showing desired and actual values and the errors of each cluster-space variable can be shown. Multiple runs can be combined into a batch, with the errors displayed in a surface plot. Both of these methods of analysis are shown in the examples in the next section.

### 3.2 Example Simulation Runs

The virtual reality display has been disabled in all of the example scripts. If the Matlab Virtual Reality Toolbox is installed on the system, the virtual reality display may be enabled by changing the `MultiRun()` argument `vr` from 0 to 1.

For simplicity, the cluster space variables $\varphi_1$, $\varphi_2$, and $\varphi_3$ are not included in either the individual run plots, or in the batch run surface plot. These variables represent the orientation of each robot, and excluding them is equivalent to stating that it is of no concern which direction the robots point in. This statement may be valid for some tasks, but not for others; if desired, the errors for these three variables can easily be enabled in the software.

Despite their exclusion from the error analysis, $\varphi_1$, $\varphi_2$, and $\varphi_3$ are included in the desired trajectory (always pointing in the direction of motion) and in the cluster space controller.

#### 3.2.1 Example 1 - Cluster Velocity and Ratio $m$

The first example (script `Example1_Velocity.m`) shows single-run results for several different velocities and ratio values. Batch results for the same range of velocities and ratios are then shown, demonstrating how a set of single runs is combined into a single result set.

**Cluster Velocity = 1, $m = 1$**

This is a slow moving cluster with the Jacobian update rate equal to the fast rate. The first two plots (Figure 3.2-1) compare the desired trajectory of the centroid of the cluster to its actual trajectory. The error is so small in this case that the lines overlap.
Figure 3.2-1: Example 1 (velocity=1, m=1): Desired and Measured trajectories in global coordinates.

A third plot (Figure 3.2-2a) shows the motion of the cluster along the trajectory. At various timesteps, the position of each robot is marked with common marker. We can see that the cluster maintains its shape as each robot precisely follows its trajectory. A fourth plot (Figure 3.2-2b) shows desired and actual cluster variables $\theta_C$, $p$, $q$, and $\beta$. The errors in $p$ and $q$ are less than 0.05 (0.25% of the desired values).

A final plot (Figure 3.2-3) shows the condition number of the $J$ and $J^{-1}$ matrices (they should always be the same, so calculating and plotting both is a good check that our
equations are correct). The condition number can vary over several orders of magnitude, so a logarithmic plot is also shown.

\[ \text{Figure 3.2-3: Jacobian condition numbers} \]

**Cluster Velocity = 2, m = 20**

With the cluster moving twice as fast and the Jacobian only being updated every 2 seconds (20 times the fast control loop period of 0.1 second), the cluster still appears to precisely follow the desired trajectory (Figure 3.2-4a). Looking more closely at the \( p \) and \( q \) values, we see that the error in these has gone from 0.05 to almost 0.2. Also, we can see the update rate reflected in the plot of actual values; these both have a ripple with a two-second period (Figure 3.2-4b).

\[ \text{Figure 3.2-4: Cluster velocity=2, m=20. Desired and measured trajectories of the cluster centroid (dark blue solid and dotted lines, overlapped with red line) and individual robot trajectories (a). Desired and measured values of } \theta_C, p, q, \text{ and } \beta \text{ vs. time (b).} \]
**Cluster Velocity = 3, m = 60**

With a higher velocity and a Jacobian update period of six seconds, we can now see visible deformation in the path taken by Robot 2 (green line in Figure 3.2-5a). This shows up in the plots of desired vs. measured p and q; the error is now around 2, about 10 times the error when m was 20. The ripple in the measured values has a period of six seconds (see p and q plots in Figure 3.2-5b), matching the update rate as expected.

![Figure 3.2-5: Cluster velocity=3, m=60. Desired and measured trajectories (a) of the cluster centroid (dark blue solid and dotted lines, overlapped with red line) and individual robot trajectories. Desired and measured values of \(\theta_C\), p, q, and \(\beta\) vs. time (b).](image)

The Jacobian condition number is only slightly higher (Figure 3.2-6), since that relates to the overall shape of the cluster, not how closely it matches the desired shape. We do see the six-second ripple in this plot, of course; the Jacobian is constant for that length of time.

![Figure 3.2-6: Jacobian condition numbers](image)
Batch Results

In any given batch, two parameters are varied while the remaining parameters are held constant. The update period is always one of the variable parameters; the fastest rate of 10 Hz (corresponding to \( m = 1 \)) is always included. The other independent variable may be the cluster velocity, \( \beta \), or \( \omega \). The user specifies a vector for the update periods (starting with 0.1 seconds), and another vector for the second variable parameter. Scalars are specified for the remaining two constant parameters.

Within a batch, the set of simulations at the 10 Hz update rate is considered the “baseline”: For each value of the second independent variable, for each of the nine cluster space variables, the error is calculated as the maximum difference between the desired and measured values. Errors at each update rate are divided by this baseline, normalizing the baseline to exactly one and allowing comparison of errors between variables whose absolute values are very different.

The two variable parameters are used as the X and Y axes of a surface plot, with the normalized error for the corresponding test run used for the Z axis (see Figure 3.2-7). Each of the nine cluster space variables is used to create such a plot. The update period is always presented as the X-axis, increasing to the right, with the second independent variable as the Y-axis increasing to the left. These plots will always intersect the XY plane along the Y-axis, because the 10 Hz baseline values always have a normalized value of 1.

Below (Figure 3.2-7) we see selected plots for \( X_c \), \( p \), \( \beta \), and a combined error plot, produced from a test run where velocity varied from 1 to 3 and update period varied from 0.1 to 6 seconds. The plot for \( X_c \) is nearly flat; the error in \( X_c \) does not increase significantly regardless of update rate or velocity. The error for \( p \) does not increase for update rates less than 1.5 seconds (\( m=15 \)), then begins increasing linearly with update rate, so that an update rate of 6 seconds produces an error 5 to 6 times the baseline error. The error for \( \beta \) is erratic, but small; the highest value is about 1.8 times the baseline error.
The combined plot takes the maximum relative error at each point across all nine cluster space variables; Figures 3.2-7(a) through (c) show three of these plots. In Figure 3.2-7(d), the first three plots have been combined by taking the maximum normalized error at each point. We can see that for update rates over two seconds, the error for $p$ dominates. Below two seconds, the peaks in the $\beta$ error plot can be seen, appearing much smaller on the same scale as the $p$ error plot. This combined plot is the primary tool for evaluating the effect of multirate control on the ability of the cluster space controller to track a trajectory.

Figure 3.2-7: Surface Plots. Each cluster space variable has a surface plot showing the normalized error vs. the update period and velocity. (a), (b), (c) are the errors for $X_C$, $p$, and $\beta$, respectively. (d) combines the plots for each variable into a single plot of the maximum normalized error at each point.
A final plot produced for each batch shows the average condition number for the Jacobian during the test run. This is plotted with a logarithmic z-axis, because the value can vary over a wide range; unlike the cluster space variables, condition numbers are not normalized. In Figure 3.2-8 below, we see that condition numbers range from 79 to 81.

![Average Matrix Condition Number](image)

*Figure 3.2-8: Surface plot of the maximum condition number of the Jacobian matrix encountered during each test run.*

### 3.2.2 Example 2 - Angle $\beta$

The next example demonstrates the effect of the cluster angle $\beta$, which is the angle between the two sides of length $p$ and $q$. As this angle nears 180°, the cluster begins to form a straight line rather than a triangle, which also causes the Jacobian matrix to approach singularity.

With $\beta$ set to 176° and cluster velocity equal to one, the cluster travels with the three robots nearly aligned, as shown by the marker positions in Figure 3.2-9a. The error in $p$ and $q$ is approximately 0.1 (Figure 3.2-9b) units at the fast update rate of 0.1 sec, compared with an error of only 0.04 at $\beta = 135^\circ$ (see Figure 3.2-2b). Also, the Jacobian condition number is now 184.5 (Figure 3.2-9c), up from 80.1 (Figure 3.2-3).
Figures 3.2-9: Results of a single run with $\beta=176^0$ and cluster velocity equal to one.

Increasing $\beta$ further to $178^0$ increases the $p$ and $q$ error to 0.2 (Figure 3.2-10a) and the condition number to 488, but further increases in $\beta$ to $179^0$ and $179.8^0$ (and condition number to 1700 and 40000) do not increase the $p$ and $q$ errors any further (Figures 3.2-10b and 3.2-10c).

Figure 3.2-10: Errors in $p$ and $q$ at $\beta=178^0$, $179^0$, and $179.8^0$, respectively.

Figure 3.2-11a shows the maximum error for a batch run with $\beta$ varying from $176^0$ to $179.8^0$, and the update period from 0.1 second to 4 seconds ($m$ from 1 to 40). At $176^0$, the error at $m = 40$ is only twice the error at $m = 1$. As $\beta$ increases, this multiple starts rising, to five at $178^0$, 10 at $179^0$, and 20 at $179.8^0$. Figure 3.2-11b shows that the condition number increases rapidly as $\beta$ gets closer to 180 degrees. This suggests that the increase in error due to the ratio $m$ may be related to the condition number of the Jacobian.
3.2.3 Example 3 - Position Measurement Error as Noise in $R$

To demonstrate the effect of inaccurate position measurement, we rerun the case of $\beta = 135^\circ$, velocity = 1, and $m = 1$ with noise enabled. This causes a random error to be added to each of the three global XY coordinates in the $R$ vector before conversion to the cluster-space $C$ vector. As shown in Figure 3.2-12a below, the noise causes the maximum error in $p$ and $q$ to increase by a factor of ten, from 0.04 to nearly 0.5.

If we increase the ratio $m$ to 60 (a six-second update period) while noise is enabled, the error becomes irregular (Figure 3.2-12b).
If we try to make $\beta$ any larger, however, the noise will cause the robot cluster to approach a straight line, creating a singular Jacobian and crashing the Matlab script. This results in an error that looks something like this:

Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND = 2.331328e-019.

To avoid the singularity of the straight-line configuration, we can use the ShapeMode option, demonstrated in Example 4 below.

### 3.2.4 Example 4 - Shape Mode for Reduced Condition Number

Until now, all simulations have been run using ShapeMode = 1 (Figure 3.1-1a). If we reposition the robots using ShapeMode = 2 so that Robot 2 is in the lead (see Figures 3.1-1b and 3.1-3), the straight-line configuration is no longer singular, and adding noise will not cause the code to halt. We rerun the case with $\beta = 176^\circ$, velocity = 1, and $m = 1$, shown in Figures 3.2-9a through c, to demonstrate this. Using this alternate cluster shape definition, the Jacobian condition number is around 95 (not shown), as opposed to 184 using the default robot assignment. We can even set $\beta$ to exactly 180 degrees, with error in $p$ and $q$ still less than 0.5, and the condition number remaining near 95 (Figure 3.2-13).

![Figure 3.2-13: Example 4: $\beta = 180$ degrees, using ShapeMode=2 (noise enabled).](a) (b) (c)

To show the improved behavior near $\beta = 180^\circ$, we run the batch from Example 2 (Figure 3.2-11), adding a final element of 180 to the end of the beta vector, extending the update rate vector to $m=80$, and enabling noise. The maximum error now increases by less than.
50% even with \( m \) up to 40 (Figure 3.2-14a), and the condition number remains at about 95 instead of rising exponentially (Figure 3.2-14b).

This last result indicates that a configuration which is singular using one cluster-space definition may not be singular using an alternate definition. Choosing a different definition may therefore reduce computational load by allowing a higher value of \( m \) and a slower Jacobian update rate.

### 3.2.5 Example 5 - Rate of Change of \( \beta \): \( \omega \)

Cluster space control allows for the pose of the cluster to change during a mission. For instance, the robots may need to move closer together to fit through an opening. The Jacobian is a function of \( C \), so changes to the pose can be expected to affect the Jacobian just as changes to the cluster’s global position and orientation affect it.

To see the effect of multirate control when the cluster pose is changing, \( \beta \) cycles sinusoidally +/-70° from its initial value. The rate (in Hz) that \( \beta \) cycles at is called \( \omega \). \( \omega \) is normally set to zero, keeping \( \beta \) (and the cluster pose) constant. Figure 3.2-15 shows VRML images from a test run with \( \omega = 0.05 \text{ Hz} \) and \( \beta = 90^\circ \), so that \( \beta \) cycles between 10° and 160° and back over a period of 20 seconds.
Figure 3.2-15: Example 5, VRML display. With $\omega$ enabled, $\beta$ cycles between $160^\circ$ (a) and $10^\circ$ (b) as the cluster follows the trajectory.

Figure 3.2-16a shows the paths taken by the robots in the preceding test run. The marks on the paths show the change in cluster pose from $90^\circ$ (crosses) to $10^\circ$ (circles and diamonds) and near $160^\circ$ (pluses and squares). Figure 3.2-16b shows that the error in $p$ and $q$ remains at one unit or less.

Figure 3.2-16: Example 5, $m = 1$. With $\omega$ enabled and the Jacobian update rate at the fastest rate of 10 Hz, the cluster pose changes as $\beta$ cycles between $90^\circ$ (marked with 'x') $160^\circ$ ('o') and $10^\circ$ (squares) (a), resulting in limited error in $p$ and $q$ (b).
When multirate control is added, slowing down the Jacobian update by a factor of \( m = 20 \), the error becomes much larger, as shown in Figure 3.2-17.

![Figure 3.2-17: Example 5, m = 20. When the Jacobian update rate is slowed down to 0.5 Hz (m = 20), the cluster pose cannot properly track the trajectory (a), resulting in a larger error in p and q (b).](image)

### 3.3 Estimate of Computational Savings

To estimate the amount of computation saved by multirate control, we traverse the control loop shown in Figure 2.2-1, inspecting the calculations performed within each block (Table 3.3-1). A direct count of calculations for the N=9 implementation used in this thesis is made, and the manner in which the count will scale with N is determined, using \( O(N) \) (“order N”) notation. These calculations are then assigned to one of two categories: those that may occur only on one of every \( m \) loop iterations ("multirate"), and those that must occur on every iteration of the fast loop ("non-multirate").

For simplicity, it is assumed that all math operations have equivalent cost: addition, subtraction, multiplication, division, exponentials, and trigonometric functions are each counted as a single operation. It is also assumed that all redundant operations have been combined to produce the most efficient implementation.
Figure 3.3-1: Repeat of Figure 2.2-1. Three-robot cluster space implementation. Solid lines are the fast control loop (10 Hz); dotted lines are calculations dependent on \( R \) which may be done at a slower rate.

The Jacobian \( J \) and its inverse \( J^{-1} \) are each calculated independently; matrix inversion is not used. Examining the Matlab code for calculating \( J \) (\textit{three_bots_centroid_jacobian_matrix_beta_atan.m}) leads to a total of 32 operations. The code for \( J^{-1} \) (\textit{three_bots_centroid_inv_jacobian_matrix_exact.m}) is much more complex; possibly thousands of operations are involved. However, there is a great deal of repetition of large blocks of calculations; reusing these code blocks results in approximately 405 operations.

The scaling with \( N \) of operations in \( J \) and \( J^{-1} \) can vary widely, depending on the manner in which the kinematic equations are defined. If each forward equation refers to very few global-space variables, the Jacobians will become sparser as \( N \) increases, and the count of operations to compute them increases as \( O(N) \). At the other extreme, if each forward kinematic equation depends on every other global-space variable, then the number of nonzero elements in \( J \) and \( J^{-1} \) is \( N^2 \), and the count of operations increases with \( O(N^2) \).

The PID controller takes the differences of two pairs of \( N \)-length vectors, performs two \( N \)-length vector integrations, three \( N \)-length element-by-element vector multiplications, and sums the three resulting vectors. It is therefore always an \( O(N) \) operation, with 99 operations in the case of \( N=9 \), and must be done at the fast loop rate.
\( W \) is calculated from equation 3-1(below), and is the product of three \( N \times N \) matrices, an \( O(N^3) \) operation if the Jacobians are fully populated. If the Jacobians are sparse, then the fact that \( M \) is a diagonal matrix will make the calculation \( O(N) \). Regardless of the order, \( M \) is constant, so the entire calculation of \( W \) can be done at the same slow rate as the Jacobians.

If the Jacobians were sparse, \( W \) will also be sparse, with the number of nonzero elements proportional to \( N \), so the product of \( W \) and the \( N \)-length vector \( F_{\text{PID}} \) to form \( F_W \) (equation 3-2; see below) will also be \( O(N) \). If the Jacobians are fully populated, equation 3-2 will be \( O(N^2) \). Because \( F_{\text{PID}} \) changes rapidly, this last calculation is not multirate.

\[
W = J^T M J^{-1} \quad (3-1) \quad F_W = WF_{\text{PID}} \quad (3-2)
\]

\[
\alpha = (J^T)(B - [MJ^{-1}\dot{J}]) \quad (3-3) \quad F_{\alpha} = \alpha \dot{\lambda} \quad (3-4)
\]

The arguments that apply to equations 3-1 and 3-2 also apply to 3-3 and 3-4. \( B \) is also a constant diagonal matrix, and \( \dot{J} \) (the time derivative of \( J \)) will be sparse if \( J \) is sparse, making the calculation of \( \alpha \) and \( F_{\alpha} \) (eqns 3-3 and 3-4) both \( O(N) \). If \( J \) is fully populated, they are \( O(N^3) \) and \( O(N^2) \), respectively. \( \alpha \) is multirate but \( F_{\alpha} \) is not.

The summation of \( F_W \) and \( F_{\alpha} \) to form \( F_C \) is \( O(N) \), but not multirate. Torque \( \tau \) and cluster-space velocity \( \dot{C} \) will both be \( O(N) \) if \( J \) is sparse, and \( O(N^2) \) otherwise; neither product is multirate. The forward kinematics equations (three_robot_centroid_forward_kin_v3.m) which lead to dense Jacobians will have each of the \( N \) equations depend on each of the \( N \) global-space variables, so will themselves be \( O(N^2) \); simpler kinematics which lead to sparse Jacobians will be \( O(N) \). Although the assumption of a slowly-changing \( R \) vector is central to the multirate technique, \( C \) must be updated with every loop iteration for use in the PID controller. These counts are summarized and totaled in Table 3.3-1 below.
Table 3.3-1: Operations for a Single Control Loop Iteration

<table>
<thead>
<tr>
<th>Calculation</th>
<th>Operation Scaling</th>
<th>Dense Jacobian</th>
<th>Sparse Jacobian</th>
<th>Operation Count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Multirate</td>
<td>Non-Multirate</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>O(N)</td>
<td>O(N²)</td>
<td>O(N)</td>
<td></td>
</tr>
<tr>
<td>J⁻¹</td>
<td>O(N)</td>
<td>O(N)</td>
<td>O(N)</td>
<td></td>
</tr>
<tr>
<td>PID</td>
<td>O(N)</td>
<td>O(N)</td>
<td>O(N)</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>O(N)</td>
<td>O(N)</td>
<td>O(N)</td>
<td></td>
</tr>
<tr>
<td>FW = WFW⁻¹</td>
<td>O(N²)</td>
<td>O(N)</td>
<td>O(N)</td>
<td></td>
</tr>
<tr>
<td>α</td>
<td>O(N³)</td>
<td>O(N)</td>
<td>O(N)</td>
<td></td>
</tr>
<tr>
<td>FA = αFR</td>
<td>O(N³)</td>
<td>O(N)</td>
<td>O(N)</td>
<td></td>
</tr>
<tr>
<td>FC = FW + FA</td>
<td>O(N)</td>
<td>O(N)</td>
<td>O(N)</td>
<td></td>
</tr>
<tr>
<td>τ = J⁺FC</td>
<td>O(N³)</td>
<td>O(N)</td>
<td>O(N)</td>
<td></td>
</tr>
<tr>
<td>ĉ = Jĉ</td>
<td>O(N³)</td>
<td>O(N)</td>
<td>O(N)</td>
<td></td>
</tr>
</tbody>
</table>

Highest Order: O(N³)  O(N³)  O(N)  O(N)  Total: 1324  720  2044

Under multirate control, every \( m \) loop iterations will require one execution of the multirate operations and \( m \) executions of the non-multirate operations. For the dense Jacobian case, as \( N \) increases and the \( O(N^3) \) calculations dominate, multirate control can reduce the total calculations per loop by a factor approaching \( m \). In an ideal case where \( m \) approaches \( N \), the system would effectively be reduced to \( O(N^2) \). For the sparse Jacobian case, the non-multirate calculations will scale with \( N \) and the maximum reduction in operations will be the ratio of non-multirate operations to the total number of operations per loop. In the sample cluster used in this thesis, a value of \( m=10 \) would reduce the average operations per loop from 2044 to 132+720=852, a reduction of 65%. 
4 Summary

The simulation described in this paper provides a starting point for investigating the effects of multirate control, where the Jacobian is updated at a slower rate than the main control loop. The error between the desired and actual trajectories can be displayed graphically as a function of the update rate and one of several other variables: cluster velocity, cluster shape, and rate of change of shape. A single global trajectory can be defined in cluster space in two different ways using the ShapeMode setting. Also, random error can be added to the position measurements, to simulate the inherent error in a real-world system.

The most important result from this project is the observation that the ratio $m$, when limited by proximity of a singularity in one cluster space definition, may not be so limited using a different cluster space definition. Another result is that identical global trajectories can be achieved even if the underlying cluster space definitions are different. Together, these results suggest that if two different cluster space definitions can be found that have sufficiently separated singularities, any global trajectory can be successfully followed by a multirate controller that allows for changing the underlying cluster space definition.

It is shown that the expected computational savings from multirate control is highest for systems for which the Jacobians are dense. In such systems the computational load scales with the cube of the number of robots, and multirate control can potentially reduce the load by a factor approaching $m$. As Jacobian sparsity increases, the computational savings offered by multirate control is reduced.

4.1 Future work

The results of this project suggest a number of avenues that could be further investigated using this simulation software. Three such avenues are described in detail below.
4.1.1 Hardware Implementation

Implementing the multirate features in an actual cluster space-controlled hardware system would be a logical next step. The process of such an implementation would validate the simulation and suggest new features. Also, the actual computational savings of multirate control could be observed.

Some of the examples show values of $m$ (20 to 60) which are much higher than any of the cases cited in the literature (section 2.3), where $m$ has a maximum values of 11 (and is usually much less than that). This suggests that the fast loop rate of 10 Hz is faster than it needs to be. Certainly, before implementing multirate control, the fast loop rate would be reduced until the accuracy was the minimum necessary for the task. This would have the effect of reducing $m$ even if the Jacobian update rate remains unchanged. For instance, if the fast update rate could be decreased to 2 Hz, $m$ would be reduced by a factor of 5.

4.1.2 Different Choices of Cluster Shape Variables

The simulation uses a particular choice of cluster shape variables, where the centroid of the triangle formed by the three robots is the cluster space origin, and the triangle is defined by side-angle-side. There are many different ways these shape variables could be chosen (see Figure 4.1-1). Just as we have seen that simply renumbering the robots within our centroid-base scheme can avoid a singularity (section 3.2.4), it may be that a different shape variable scheme offers a performance advantage (greater accuracy, higher $m$) under multirate control.

Figure 4.1-1: Alternate choices of cluster shape variables. Altitude-Base-Angle (a), Angle-Side-Angle (b), Side-Side-Side (c). Note that the cluster origin is placed on a robot, rather than the triangle centroid. [12]
4.1.3 Transparent Shape Definition Change

It was demonstrated that a single global trajectory, defined in two different ways in cluster space, can have very different sensitivity to multirate control. In this implementation, the definition must be chosen at the start of the simulation, and cannot be changed during the simulation. This would be a problem if the desired trajectory encounters singularities in both definitions; neither one would allow completion of the task.

For this reason, it would be useful for the control system to be able to change the shape definition “on-the-fly”, that is, while a simulation is in progress. This would require reassigning the designations (Robot 1, Robot 2 and Robot 3) to different physical robots, and changing the trajectory definition, while maintaining the definitions presented to the operator. Further, once this ability is implemented, it would be desirable for the control system to detect when such a switch is necessary, and make the change in a way that is transparent to the operator.
5 References


Appendix: Software Description

The files associated with this thesis contain both the multirate simulation and the unmodified original single-rate simulation on which the multirate simulation is based. The differences between the two versions are documented here.

File Listing

New Files

These files are not in the original simulation, but have been added to the multirate simulation.

*cluster_three_ideal_robots_novr.mdl*

Identical to cluster_three_ideal_robots.mdl, except Virtual Reality display has been removed to speed up the simulation and allow it to run on systems without the Virtual Reality Toolbox.

*Example1_Velocity.m, Example2_Beta.m, Example3_Noise.m, Example4_ShapeMode.m, Example5_Omega.m*

These example scripts are described in section 3.2.

*GetJacobianConditionNumber.m, GetInvJacobianConditionNumber.m*

Calculates the condition number of the Jacobian and inverse Jacobian matrices, and stores them in global variables.

*MultiResult.m*

Displays a set of surface plots, where each $x,y$ point of the surface represents a single simulation run.

*MultiRun.m*

Runs a set of simulations, where the ratio $m$ and one other parameter are each varied.

*NormalizeAngles.m*

Angles measured from the dynamic robot model may be offset from the desired trajectory by $+/- \ 2\pi$; this function adjusts the measured angle by $2\pi$. 
SetSimGlobalVariables.m
Declares all of the global variables used in the multirate simulation. Called by the Simulink mdl files in InitFcn.

SingleRunResult.m
Displays a set of plots, showing details about a single simulation run.

SmoothTurnTrajectoryGenerator.m
Creates the trajectory used in the multirate simulation.

three_robot_centroid_AddNoise.m
Adds noise to the $x$ and $y$ of each robot’s global position.

three_robot_centroid_initial_positions.m
Computes the global positions of robots 1, 2 and 3 based on the cluster centroid global position, $p$, $q$, and $\beta$. Used to initialize robot positions to desired cluster pose, to avoid large transient error at the start of simulation.

**Changed Files**
These files are in both the original and the multirate simulations, but have been changed to support multirate control.

cluster_three_ideal_robots.mdl
Simulink model implementing nonlinear cluster space control of three-robot cluster. Modified to implement multirate control.

three_bots_centroid_inv_jacobian_matrix_exact.m
Calculates the inverse Jacobian of the three-robot cluster. Modified to implement multirate control.

three_bots_centroid_jacobian_matrix_beta_atan.m, three_bots_jacobian.m
Calculates the Jacobian of the three-robot cluster. Modified to implement multirate control.
Deleted Files

These files are in the original simulation, but do not appear in the multirate simulation.

*a_result_generation_script.m*

Displayed results of original single-rate control Simulink model; heavily modified for multirate implementation and the name was changed to *SingleRunResult.m*.

*three_bots_inv_jacobian.m*

Not used in multirate implementation.

*getting_j_dot.txt, getting_j_dot.html, getting_j_dot.mn*

Used to obtain code for derivative of Jacobian; no longer needed.

*three_bots_centroid_jacobian_dot_matrix.asv, get_beta_dot_dot.asv, three_bots_cluster_level_obstacle_avoidance_ellipse_v3.asv*

Autosave files from Matlab editor; not needed and deleted.

Unchanged Files

These files appear in both simulations, and have not been changed.

*alternate_objects.m, cluster_three_ideal_robots_sfun.mexw32, get_beta_dot_dot.m, get_heading_angle_from_vel.m, heading_error_conditioner.m, in_line_trajectories.mdl, inv_jacob.mat, joystick_controller_output_position.mdl, minvolellipse.m, obstacle_avoidance_multiobstacle_v2.m, robot_global_local_transformation.m, three_bots_centroid_jacobian_dot_matrix.m, three_bots_cluster_level_obstacle_avoidance_ellipse_v3.m, three_robot_centroid_forward_kin_v3.m, three_robot_centroid_inverse_kin_v3.m, three_romeos_and_obstacle2.wrl, trajectory_generators.mdl*

In cvs\directory: entries, entries.extra, entries.extra.old, entries.old, repository, root

Instructions

Associated with this thesis is a Matlab simulation implementing multirate control of a three-robot cluster. The code and simulation were developed on the Windows Server 2003 machines in the Santa Clara University Engineering Design Center, using Matlab R2008b. In addition, the simulation has been run successfully on Windows 7 Professional using Matlab R2010a Student Version, with Virtual Reality disabled. The
complete simulation is contained within a single compressed file. The original simulation, from which the multirate simulation was developed, is also included.

To operate the simulation, expand the contents of the compressed file into a directory, and set the directory as the Matlab Working Directory.

The function/script *MultiRun.m* will run one or more simulations in a batch. Velocity, $\beta$, $\omega$, and Update Rate are vector arguments; for a given batch, the update rate and at most one other parameter can have more than one element. *ShapeMode*, *vr*, and *noise* are flags which enable or disable the respective feature.

There are five example scripts included (see section 3.2). Each script will call *MultiRun.m* with arguments for a single run and display the results. There are also comment lines with vector arguments in some of the examples; uncommenting these lines will cause the example to run a batch, and display a surface plot.