ENPM667
Project - I

TECHNICAL REPORT

ON

Planning and Control of Ensembles of Robots with Non-holonomic Constraints

Rachith Prakash (116141468)
Janakiraman Kirthivasan (116349086)

University of Maryland, College Park MD

Contents

List of Figures

Abstract

One of the new approaches in building an intelligent decision-making system is usage of swarm robots. These multi-robot systems are emerging as a more efficient systems in the field of artificial swarm intelligence and in agriculture due to their desired collective behavior interacting with the surrounding and other robots in solving various problems based on the inputs.

In this technical report, development of a robot ensemble with non-holonomic constraints using a control law to achieve a desired position, orientation and shape of the formation is being implemented. The robot swarm is controlled through individual team members along with minimal knowledge of the ensemble state. Furthermore, the control of the robot swarm is independent of the number of robots in the team that ensures the system is stable to failures in individual members. In addition, inter-robot collision avoidance, motion planning of the ensemble is detailed in the report. In order to achieve a desired distribution of a defined number of robots a decentralized control law safe from inter-robot collisions has been derived. The results of the algorithms are simulated for a differential drive robot using MATLAB software tool.

Chapter 1

INTRODUCTION

Study and research of robot swarms has been an area that is vastly developing due to technological advancements in terms of sensing, computing etc. and its various applications in fields such as security and defense, monitoring of the environment, search and rescue operation etc. Hence, effective control strategies for the robot swarms is being developed and is necessary for execution of complex tasks.

Various methodologies have been adapted to control the formation of robots such as control through formation graphs, controlling by maintaining a rigid virtual structures within the team, leader follower architecture etc. However, these methods have drawbacks of sensitivity to failures of individual members and requirement of re-ordering of the robots.

This report details one of the methodology developed by Michael and Kumar [2] to control the orientation and shape of the team of mobile robots which are independent of the count and ordering of the team. The principle of the methodology is modelling of formation using an abstract state which describes the shape and pose of the entire team while being independent of all team members. Thus, the algorithm holds good for teams of varying size. Also, the abstract state is decoupled, which makes the control design effective and this is further detailed in the report. The report details the application of this algorithm to non-holonomic robots taking into consideration the avoidance of collision along with motion planning.

Chapter 2

BACKGROUND

This report consists of previous work of [1] Belta and Kumar (2004), [4] Michael et al. (2006), [3] Michael et al. (2007) and [2] Michael and Kumar (2008). In the reference research papers, an abstraction map is used to transform the high-dimensional state space into a smaller, tractable state space which captures only the position, orientation, and shape of the formation. The main advantages of this abstract representation are: (a) its dimension is independent of the number of robots in the team; (b) it lends itself to planning in a lower-dimensional space; and (c) minimum communication between robots.

2.1 Problem Formulation

The state space of the N-robot system is constructed by creating N copies of Q_i , the state space of the i^{th} robot:

$$Q = Q_1 \text{ x } Q_2 \text{ x } \ldots\ldots \text{ x } Q_N \tag{2.1}$$

Given a large number of robots evolving on the configuration space Q also considered as manifold in this report, we want to be able to solve motion-generation/control problems on a smaller dimensional space, which captures the essential features of the group, according to the class of tasks to be accomplished. We want the dimension of the control problem to be independent of the number of agents and independent of the possible ordering of the robots. These requirements will provide good scaling properties and control laws which are robust to individual failures.

We also need to make sure that, after solving the task on the small dimensional space, we can go back and generate control laws for the individual agents. All of these ideas lead to the following definition.

$$\phi : Q \mapsto M, \quad \phi(q) = x \tag{2.2}$$

where ϕ is the surjective submersion mapping from higher-dimensional state $q \in Q$ to lower-dimensional abstract state $x \in M$. Map ϕ is called abstraction as it is invariant to permutations of the robots and the dimension n of M is not dependant on the number of robots N. M is also called abstract manifold and x is called the abstract state.

Consider N kinematically controlled robots with states q_i belonging to manifold Q_i and control space U_i. For planar fully actuated robots, the states are position vectors $q_i \in Q_i = \mathbb{R}^2, i = 1, \ldots\ldots, N$ with respect to world frame {W} and the controls $u_i \in U_i = \mathbb{R}^2$ as follows:

$$\dot{q}_i = u_i \tag{2.3}$$

In addition, M is considered to have a product structure of the form

$$M = G \text{ x } S, \quad x = (g, s), \quad \phi = (\phi_g, \phi_s) \tag{2.4}$$

where G is a Lie group. $g \in G$ defines the position and orientation of the team of robots in the world frame W and is called the group variable. $s \in S$ defines the shape of the team and is called the shape variable. Thus, there should be a control-suited description of team of robots x in terms of group variable g of a local frame, which captures the dependence of the team on the world frame W, and a shape s, which is decoupled from g, and hence an intrinsic property of the formation. Thus group G is left invariant i.e. for any arbitrary element $\bar{g} \in G$, the map ϕ satisfies the following

$$\phi(q) = (g, s) \Longrightarrow \phi(\bar{g}q) = (\bar{g}g, s) \tag{2.5}$$

where $(\bar{g}g)$ represents the block diagonal action of the group elements $\bar{g}$ on the configuration $q \in Q$ and $\bar{g}g$ represents the left translation of g by $\bar{g}$ using composition rule on group G. In the case of planar robots, $\bar{g}g$ would represent a rigid displacement of all of the robots by $\bar{g}$. However, this displacement will not change the shape s of the team of robots. Thus, the control laws based on the abstract state $x = (g, s)$, will be invariant to pose of the world frame W. Thus, instead of designing high-dimensional behaviors X_Q, we will be able to describe collective behaviors in terms of time-parameterized curves, X_M on the lower dimensional abstract manifold M.

Next the author has defined an abstract behavior. What is an abstract behavior? Any vector field $X_M \in TM$ i.e. tangent space of M is called an abstract behavior. Now, $d\phi$ represents the differential also called tangent of the map ϕ. Since, ϕ is a submersion, it ensures the surjectivity of $d\phi$ at any point $q \in Q$. This would guarantee the existence of vector fields X_Q mapped to any abstract behaviour X_M i.e all behaviors $\dot{x}$ in X_M(co-domain) is covered by behaviors $\dot{q}$ in X_Q(domain).This would mean that some behaviors in manifold Q can be seen in abstract manifold M and some behaviors cannot be seen in M i.e some behavior $X_Q \in TQ$ is mapped to a non-zero $X_M \in TM$ and some might not. Those behaviors $X_Q \in TQ$, which are mapped to X_M are called detectable behaviors. Behaviors which are not mapped to X_M are called non-detectable behavior.

2.1.1 Goal

The goal of the report is to generate individual control laws $\dot{q} \in X_Q$ which are mapped to desired abstract (collective) behaviors ($\dot{x} \in X_M$), i.e., wisely chosen low-dimension descriptions. Therefore, individual motions which cannot be captured in M are not allowed, because this would be a waste of energy.

Thus the problems that are addressed and solved are as follows:

Problem: Determine physically meaningful formation abstraction ϕ, abstract behaviors X_M, and corresponding individual robot control laws $u_i \in X_Q$ satisfying the following requirements:

1. the abstract state x is stationary if and only if all the robots q_i are stationary;

2. the abstract manifold M has a product structure and satisfies the left invariance property;

3. the control systems on group G and shape S are decoupled i.e. independent of each other;
4. if the state x of the abstract manifold is bounded, then the state of each robot q_i is bounded;
5. monotonic convergence of the abstract state;
6. inter-collision avoidance.

The first 4 points are addressed in this chapter and the last 2 along with the minimum-energy condition is addressed in CHAPTER 3 and CHAPTER 4. In addition to the requirements explicitly formulated in the above **Problem**, it is desired that the energy spent by the individual robots to produce a desired abstract behavior be kept to a minimum. Thus, control inputs that satisfy a minimum-energy constraints are obtained. Also, the amount of inter-robot communication in the overall control architecture should be limited.

2.2 Approach to Solve the Problem Formulated in 2.1.1

In this section a solution to the above **Problem** 2.1.1 is characterized. Let us assume that the abstract manifold M has a product structure $M = G$ x S. If D is a distribution of a manifold M of dimension n, then D_x, is the subset of tangent space of abstract manifold M and is given by $D_x \subset TM, \forall x \in M$. Also, D can be interpreted as $D_x = \text{span}\{X_{1_{|\text{x}}}, X_{2_{|\text{x}}}, ..., X_{r_{|\text{x}}}\}$, where r is the number of linearly independent vector fields X_r in the tangent space of manifold M around point x. Let Ω_g be the co-distribution spanned by the differential forms obtained by differentiating each component of ϕ_g. Similarly, Ω_s will be the co-distribution determined by ϕ_s. Let Δ_g and Δ_s denote the corresponding annihilating distributions, i.e. the distributions that maps Ω_g and Ω_s to zero is given by,

$$\Omega_g(\Delta_g) = 0, \quad \Omega_s(\Delta_s) = 0 \tag{2.6}$$

Let $\bar{\Delta_g}$ be any distribution such that $\bar{\Delta_g} + \Delta_g = TQ$ and $\dim \bar{\Delta_g} + \dim \Delta_g = \dim Q$. Similarly $\bar{\Delta_s}$ be any distribution such that $\bar{\Delta_s} + \Delta_s = TQ$ and $\dim \bar{\Delta_s} + \dim \Delta_s = \dim Q$. Then detectable behavior,

$$X_Q \in \bar{\Delta_g} \tag{2.7}$$

guarantees that, on the abstract manifold $x = (g, s)$, g changes in time whenever q does. Similarly

$$X_Q \in \bar{\Delta_s} \tag{2.8}$$

corresponds to a change in shape variable s. Thus, the set of detectable behaviors is given by $\bar{\Delta_g} + \bar{\Delta_s}$. Thus

$$X_Q \in \bar{\Delta_g} + \bar{\Delta_s} \tag{2.9}$$

Main idea is choosing $X_Q \in TQ$ such that any change in the behavior of $\dot{q} = u$ in Q must be detectable in M and since any changes in the abstract manifold M is mapped back to behaviors in Q (ϕ is surjective), data loss can be minimized. Hence, the system is forbidden to move on a leaf $\phi = constant$ (since, $d\phi = 0$ in this case), which is equivalent to any motion that cannot be observed on abstract manifold M iff (2.9) is satisfied.

Next, to formulate the decoupling condition between the control of group G and the shape of S of manifold M, we first require that the distributions $\bar{\Delta_g}$ and $\bar{\Delta_s}$ be independent, i.e., $\bar{\Delta_g} \cap \bar{\Delta_s} = 0$, where 0 denotes the zero vector field. Then the decoupling condition is satisfied if the co-distribution corresponding to g annihilates the visible motion corresponding to s and the other way around. Explicitly,

$$\Omega_g(\bar{\Delta_s}) = 0, \quad \Omega_s(\bar{\Delta_g}) = 0 \tag{2.10}$$

Now, in order to verify this, let us take $g = \phi_g(q)$. When g is differentiated, $\dot{g} = d\phi_g \dot{q}$ is obtained. If $\dot{q}$ is detectable (satisfies (2.9)), it can be written as $\dot{q} = A_s u_s + A_g u_g$, where A_g and A_s are matrices whose columns span $\bar{\Delta_g}$ and $\bar{\Delta_s}$ respectively. Now, u_s does not affect $\dot{q}$ only when changes in u_s does not affect the dependence of $\dot{q}$ on u_g i.e. $d\phi_g A_s = 0$, which in turn means, $\Omega_g(\bar{\Delta_s}) = 0$. u_g and u_s separately control g and s and will be the actual controls for group and shape.

Thus, given a vector field $X_M \in TM$, the set of all vector fields $X_Q \in TQ$ which maps to X_M is underdetermined as it's not a bijective mapping. Let $\dot{q}$ and $\dot{x}$ denote the coordinates of X_Q and X_M, respectively. Then

$$d\phi \dot{q} = \dot{x} \tag{2.11}$$

Here $\dot{q}$ and $\dot{x}$ denote the coordinates of the tangent space of X_Q and X_M, respectively.

Note that this equation is of the form $AX = B$ where $A = d\phi, X = \dot{q}, B = \dot{x}$. Since the equation is underdetermined, it can be solved by minimizing $l2$ norm of vector $\dot{q}$, since we are considering q to be defined in $\mathbb{R}^2$ i.e. planar.

Also, since ϕ is a submersion, more precisely ϕ is a submersion at $q_i, i = 1, 2, ..., N$ such that its differential $d\phi : X_{q_i}Q \mapsto X_{\phi(q_i)}M$ is a surjective linear map. A differential map ϕ that is a submersive at each point $q_i \in Q$ is called a submersion. Equivalently, ϕ is a submersion if its differential $d\phi$ has a constant rank equal to the dimension of M. ϕ_1, ϕ_2, ..., ϕ_n, n being the dimension of abstract state $x \in M$ are functionally independent or, equivalently, $d\phi$ is full-row rank. Hence, there exists an inverse $d\phi^+$ which is given by the right Moore-Penrose Inverse or commonly called Pseudo-Inverse (for full row-rank matrix):

$$A^+ = A^T(AA^T)^{-1} \tag{2.12}$$

Substituting these values gives

$$d\phi^+ = d\phi^T(d\phi d\phi^T)^{-1} \tag{2.13}$$

Thus the solution to minimization problem, $min_{\dot{q}} \dot{q}^T \dot{q}$ under constraint (2.11) is the least norm in Euclidean sense given by

$$\begin{aligned} \dot{q} &= d\phi^+ \dot{x} \\ \dot{q} &= d\phi^T(d\phi d\phi^T)^{-1} \dot{x} \end{aligned} \tag{2.14}$$

Since $d\phi$ is full row rank, it means $\dim row(d\phi) < \dim column(d\phi)$, hence an inverse exists for $d\phi d\phi^T$, where $d\phi^T = (d\phi_g^T, d\phi_s^T)$, $\dot{x} = (\dot{g}, \dot{s})$. (2.14) becomes

$$\begin{aligned}
\dot{q} &= (d\phi_g^T, d\phi_s^T)((d\phi_g, d\phi_s)(d\phi_g^T, d\phi_s^T))^{-1}\dot{x} \\
\dot{q} &= (d\phi_g^T, d\phi_s^T)(d\phi_g d\phi_g^T + d\phi_s d\phi_s^T)^{-1}(\dot{g}, \dot{s}) \\
\dot{q} &= (d\phi_g^T, d\phi_s^T)((d\phi_g d\phi_g^T)^{-1} + (d\phi_s d\phi_s^T)^{-1})(\dot{g}, \dot{s}) \\
\dot{q} &= (d\phi_g^T (d\phi_g d\phi_g^T)^{-1} + d\phi_s^T (d\phi_s d\phi_s^T)^{-1})(\dot{g}, \dot{s}) \\
\dot{q} &= d\phi_g^T (d\phi_g d\phi_g^T)^{-1}\dot{g} + d\phi_s^T (d\phi_s d\phi_s^T)^{-1}\dot{s} \qquad (2.15)
\end{aligned}$$

if $d\phi_g d\phi_s^T = 0$ and $d\phi_s d\phi_g^T = 0$.

This $\dot{q}$ satisfies the detectability and decoupling condition in terms of (2.9) and (2.10) i.e $\bar{\Delta_g}$ and $\bar{\Delta_s}$ are spanned by $d\phi_g^T$ and $d\phi_s^T$ respectively. Linear independence of $d\phi_g$ and $d\phi_s$ implies linear independence of $\bar{\Delta_g}$ and $\bar{\Delta_s}$ and (2.6) implies that $\bar{\Delta_g}$ and Δ_g are orthogonal. (2.10) is implied by $d\phi_g d\phi_s^T = 0$ and $d\phi_s d\phi_g^T = 0$.

In order to limit the amount of inter-robot communication in the overall control architecture, author designs an architecture where the control law of robot depends on its own state and lower-dimensional abstract state of the team from group manifold, $x = (g, s)$, as follows:

$$u_i = u_i(q_i, x) \qquad (2.16)$$

2.3 Physical Significance of Abstraction

In this section, abstraction x defined previously is explained with it's physical significance.
For an arbitrary $q \in Q$, the group part g of the abstract state $x = (g, s)$ is defined by $g = (R, \mu) \in G = SE(2)$. The shape s is modeled by characterizing the distribution of robots about their mean position. The centroid of the group in world frame $\{W\}$ is given by:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} q_i \in \mathbb{R}^2 \qquad (2.17)$$

Let us define any point r_i in the abstract space x as follows,

$$r_i = [x_i, y_i] = R^T(q_i - \mu) \qquad (2.18)$$

that satisfy

$$\sum_{i=1}^{N} x_i y_i = 0 \qquad (2.19)$$

meaning the distribution of robots in the abstract space is such that off-diagonal elements of the co-variance matrix of the distribution of the robots are zero.The parameterization R is defined by:

$$R = \begin{pmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{pmatrix} \in \mathbb{R}^2. \qquad (2.20)$$

Since we are dealing with shapes in 2-D, $s = (s1, s2)$, defined by:

$$
\begin{aligned}
s_1 &= \frac{1}{N-1}\sum_{i=1}^{N} x_i^2 \\
s_2 &= \frac{1}{N-1}\sum_{i=1}^{N} y_i^2
\end{aligned}
\tag{2.21}
$$

Since $R \in SO(2)$ is one dimensional (1-D) i.e. 1 Degree of Freedom (DOF); θ is sufficient to describe R, the dimension of the abstract manifold, $M = (g, s) = (R, \mu, s), \mu \in \mathbb{R}^2, s \in \mathbb{R}^2$ is n=5, independent of the number of robots N.

Let us now study the physical significance of the abstraction ϕ. In a planar case, the covariance matrix Σ and the inertia Tensor Γ in world frame W is defined as:

$$
\Sigma = \frac{1}{N-1}\sum_{i=1}^{N}(q_i-\mu)(q_i-\mu)^T \tag{2.22}
$$

$$
\begin{aligned}
\Sigma &= \frac{1}{N-1}\sum_{i=1}^{N}\begin{pmatrix}(q_i-\mu)_x\\(q_i-\mu)_y\end{pmatrix}\begin{pmatrix}(q_i-\mu)_x & (q_i-\mu)_y\end{pmatrix}\\
(N-1)\Sigma &= \sum_{i=1}^{N}\begin{pmatrix}(q_i-\mu)_x\\(q_i-\mu)_y\end{pmatrix}\begin{pmatrix}(q_i-\mu)_x & (q_i-\mu)_y\end{pmatrix}\\
(N-1)\Sigma &= \sum_{i=1}^{N}\begin{pmatrix}(q_i-\mu)_x^2 & (q_i-\mu)_x(q_i-\mu)_y\\(q_i-\mu)_x(q_i-\mu)_y & (q_i-\mu)_y^2\end{pmatrix}\\
(N-1)E_3\Sigma E_3 &= \sum_{i=1}^{N}E_3\begin{pmatrix}(q_i-\mu)_x^2 & (q_i-\mu)_x(q_i-\mu)_y\\(q_i-\mu)_x(q_i-\mu)_y & (q_i-\mu)_y^2\end{pmatrix}E_3\\
(N-1)E_3\Sigma E_3 &= \sum_{i=1}^{N}\begin{pmatrix}0 & -1\\1 & 0\end{pmatrix}\begin{pmatrix}(q_i-\mu)_x^2 & (q_i-\mu)_x(q_i-\mu)_y\\(q_i-\mu)_x(q_i-\mu)_y & (q_i-\mu)_y^2\end{pmatrix}\begin{pmatrix}0 & -1\\1 & 0\end{pmatrix}\\
-(N-1)E_3\Sigma E_3 &= \sum_{i=1}^{N}\begin{pmatrix}(q_i-\mu)_y^2 & -(q_i-\mu)_x(q_i-\mu)_y\\-(q_i-\mu)_x(q_i-\mu)_y & (q_i-\mu)_x^2\end{pmatrix}\\
\Gamma &= -(N-1)E_3\Sigma E_3
\end{aligned}
\tag{2.23}
$$

respectively, where $E_3 = \begin{pmatrix}0 & -1\\1 & 0\end{pmatrix}$. Here, $E_3^T = E_3^{-1} = -E_3$ and Γ and Σ have the same eigenstructure. The geometric interpretation can be done in 2 ways shown as follows.

2.3.1 Spanning Rectangle:

μ (2.17) and Γ (2.23) can be seen as the centroid and inertia tensor of the system of particles with respect to centroid and orientaion of world frame W. Now, let B denote a virtual frame with pose (position and orientation) $g = (R, \mu)$ in W. Then r_i is the expression of $q_i - \mu$ in the virtual frame B. The rotation R defines the orientation of the local frame B such that the inertia tensor of the system of points r_i in B is diagonal i.e

$$I = \sum_{i=1}^{N} \begin{pmatrix} x_i^2 & 0 \\ 0 & y_i^2 \end{pmatrix} \tag{2.24}$$

It can be clearly seen from above that $(N-1)s_1$ and $(N-1)s_2$ (using (2.21)) are the eigenvalues of the tensor I and are, therefore, measures of the spatial distribution of the robots along the axis of the local frame B.

It is interesting to note that the shape variables provide a bound for the region occupied by the robots. From (2.21), it follows that

$$|x_i| \leq \sqrt{(N-1)s_1}, \quad |y_i| \leq \sqrt{(N-1)s_2} \tag{2.25}$$

The conclusion can be stated as follows. An ensemble of N robots described by a five-dimensional (5-D) abstract variable $x = (g, s) = (R, \mu, s_1, s_2)$ is enclosed in a rectangle centered at μ and rotated by R $\in SO(2)$ in the world frame W. The sides of the rectangle are given by $2\sqrt{(N-1)s_1}$ and $2\sqrt{(N-1)s_2}$. The rectangle described by (R, u, s_1, s_2) is called the spanning rectangle.

2.3.2 Concentration Ellipsoid:

μ (2.17) and Γ (2.23) can be interpreted as sample mean and covariance of a random variable with realizations q_i. If the random variable is known to be normally distributed, then, for a sufficiently large N, μ and Γ converge to the real parameters of the normal distribution. R is the rotation that diagonalizes the co-variance, and s_1 and s_2 are the eigenvalues of the diagonalized co-variance matrix. This means that, for a large number of normally distributed robots, μ, R, s_1 and s_2 give the pose and semi-axes of a concentration ellipsoid.

For a 2-D case, the probability density function is given by:

$$p = \frac{1}{2\pi\sqrt{|\Sigma|}} \exp\left[(x-u)^T \Sigma^{-1} (x-u)\right] \quad \forall x \in \mathbb{R}^2$$

The surface or contour c for a constant probability density p is given by

$$(x-u)^T \Sigma^{-1} (x-u) = c, \quad c = -2\ln(1-p) \tag{2.26}$$

Definition of a constant probability density contour is all x's that satisfy the expression above. The ellipse in (2.26), called the equipotential or concentration ellipse, has the property that p percent of the points are inside it and can be therefore used as a spanning region for our robots, under the assumption that they are normally distributed. Thus, p percent of a large number N of normally distributed robots described by a 5-D abstract variable $x = (g, s)$ is enclosed in an ellipse centered at μ, rotated by $R \in SO(2)$ in the world frame W and with semi-axes $\sqrt{cs_1}$ and $\sqrt{cs_2}$, where c is given by (2.26).

2.3.3 Spanning Rectangle vs Concentration Ellipsoid:

The abstraction based on the spanning rectangle as defined in 2.3.1 has the advantage that it provides a rigorous bound for the region occupied by the robots and does not rely on any assumption on the distribution of the robots. The main disadvantage is that this estimate becomes too conservative when the number of robots is large. The lengths of the sides of the rectangle scale with $\sqrt{N-1}$, so for a large N the spanning rectangle can become very large, even though the robots might be grouped around the centroid μ. Thus, it would be inefficient for large number of robots.

On the other hand, the size of a concentration ellipsoid as defined in 2.3.2 does not scale with the number of robots, which makes this approach very attractive for very large N. However, it has the disadvantage of assuming a normally distributed initial configuration of the team and does not provide a rigorous bound for the region occupied by the robots. Approximately speaking, the number of robots left out of the ellipse is given by $(1-p)N$. Increasing p will decrease the number of the robots being outside but will also increase the size of the ellipsoid. Hence, an ellipsoid would be suitable for different values of N while maintaining it's size by changing the value of c based on p.

To have an idea of what is a "large" number N for which the second approach is more feasible, note that the spanning rectangle and the rectangle in which the concentration ellipsoid is inscribed are similar and the ratio is $\sqrt{(N-1)/c}$. The ratio of their areas is therefore $(N-1)/c$. For example, if $p = 0.99$, we have $c = 9.2103$ (from [5]), and the spanning rectangle becomes larger for $N \geq 11$. If $N = 100$, the area of the spanning rectangle is 10.7488 larger than the area of the rectangle circumscribing the ellipse, and only one robot might be left out of the ellipse. Thus, in our report we consider the formation to be spanned by an ellipse.

2.4 Detectable Behaviours and Decoupling of Group and Shape

In this section, under the assumption that the configuration space Q is equipped with a Euclidean metric, the author constructs detectable behaviors and decoupled control systems for group g and shape s as required by the **Problem** in 2.1.1 defined above. From (2.19), we can do the following calculations:

$$\begin{aligned}
\sum_{i=1}^{N} x_i y_i &= 0 \\
\sum_{i=1}^{N} 2x_i y_i &= 0 \\
\sum_{i=1}^{N} \begin{pmatrix} x_i & y_i \end{pmatrix} \begin{pmatrix} y_i \\ x_i \end{pmatrix} &= 0 \\
\sum_{i=1}^{N} \begin{pmatrix} x_i & y_i \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} &= 0 \\
\sum_{i=1}^{N} \begin{pmatrix} x_i & y_i \end{pmatrix} E_1 \begin{pmatrix} x_i \\ y_i \end{pmatrix} &= 0 \qquad (2.27)
\end{aligned}$$

where $E_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$.

Now $[x_i, y_i]^T = R^T(q_i - \mu)$, hence $[x_i, y_i] = (q_i - \mu)^T R$. Substituting this in (2.27), gives:

$$\sum_{i=1}^{N} (q_i - \mu)^T R E_1 R^T (q_i - \mu) = 0 \tag{2.28}$$

Let us define few matrices which are going to be used further in the report I_2, E_2, H_1, H_2, H_3:

$$\begin{aligned} I_2 &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\ E_1 &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \\ E_2 &= \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \\ E_3 &= \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \\ H_1 &= I_2 + R^2 E_2 \\ H_2 &= I_2 - R^2 E_2 \\ H_3 &= R^2 E_1 \end{aligned} \tag{2.29}$$

Now, by small manipulation, $R E_1 R^T = R^2 E_1$. Hence, the equation (2.28) becomes

$$\begin{aligned} \sum_{i=1}^{N} (q_i - \mu)^T R^2 E_1 (q_i - \mu) = 0 \\ \sum_{i=1}^{N} (q_i - \mu)^T H_3 (q_i - \mu) = 0 \end{aligned} \tag{2.30}$$

Similar transformations can be applied on the shape variables s_1 and s_2 to get its equivalent form in world frame W. (2.21) takes the form

$$
\begin{aligned}
s_1 &= \frac{1}{(N-1)} \sum_{i=1}^{N} x_i^2 \\
&= \frac{1}{2(N-1)} \sum_{i=1}^{N} 2x_i^2 \\
&= \frac{1}{2(N-1)} \sum_{i=1}^{N} x_i^2 + x_i^2 \\
&= \frac{1}{2(N-1)} \sum_{i=1}^{N} \begin{pmatrix} x_i & y_i \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} + \begin{pmatrix} x_i & y_i \end{pmatrix} \begin{pmatrix} x_i \\ -y_i \end{pmatrix} \\
&= \frac{1}{2(N-1)} \sum_{i=1}^{N} \begin{pmatrix} x_i & y_i \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} + \begin{pmatrix} x_i & y_i \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} \\
&= \frac{1}{2(N-1)} \sum_{i=1}^{N} \begin{pmatrix} x_i & y_i \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} + \begin{pmatrix} x_i & y_i \end{pmatrix} E_2 \begin{pmatrix} x_i \\ y_i \end{pmatrix} \\
&= \frac{1}{2(N-1)} \sum_{i=1}^{N} (q_i - \mu)^T R R^T (q_i - \mu) + (q_i - \mu)^T R E_2 R^T (q_i - \mu) \\
&= \frac{1}{2(N-1)} \sum_{i=1}^{N} (q_i - \mu)^T (q_i - \mu) + (q_i - \mu)^T R E_2 R^T (q_i - \mu) \\
s_1 &= \frac{1}{2(N-1)} \sum_{i=1}^{N} (q_i - \mu)^T (I_2 + R E_2 R^T)(q_i - \mu) \\
&= \frac{1}{2(N-1)} \sum_{i=1}^{N} (q_i - \mu)^T (I_2 + R^2 E_2)(q_i - \mu) \\
s_1 &= \frac{1}{2(N-1)} \sum_{i=1}^{N} (q_i - \mu)^T H_1 (q_i - \mu)
\end{aligned}
$$

Thus we have,

$$
\begin{aligned}
s_1 &= \frac{1}{2(N-1)} \sum_{i=1}^{N} (q_i - \mu)^T H_1 (q_i - \mu) \\
s_2 &= \frac{1}{2(N-1)} \sum_{i=1}^{N} (q_i - \mu)^T H_2 (q_i - \mu)
\end{aligned}
\tag{2.31}
$$

Since the rotation R is parameterized by θ the amount of rotation is restricted to $\theta \in (-\pi/2, \pi/2)$, a unique solution of

$$
\sum_{i=1}^{N} (q_i - \mu)^T R^2 E_1 (q_i - \mu) = 0
$$

is given by,

$$
R^2 = \begin{pmatrix} (q_i - \mu)^T E_2 (q_i - \mu) & -(q_i - \mu)^T E_1 (q_i - \mu) \\ 1 & (q_i - \mu)^T E_2 (q_i - \mu) \end{pmatrix} \tag{2.32}
$$

But,

$$
\begin{aligned}
R^2 &= \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \\
R^2 &= \begin{pmatrix} \cos^2\theta - \sin^2\theta & -2\cos\theta\sin\theta \\ 1 & \cos^2\theta - \sin^2\theta \end{pmatrix} \\
R^2 &= \begin{pmatrix} \cos 2\theta & -\sin 2\theta \\ 1 & \cos 2\theta \end{pmatrix} \qquad (2.33)
\end{aligned}
$$

Hence, comparing the two equations (2.32) and (2.33), and since $\tan 2\theta = \frac{\sin 2\theta}{\cos 2\theta}$ we get

$$
\begin{aligned}
\tan(2\theta) =& \frac{(q_i - \mu)^T E_1 (q_i - \mu)}{(q_i - \mu)^T E_2 (q_i - \mu)} \\
\theta =& \frac{1}{2} \arctan 2((q_i - \mu)^T E_1 (q_i - \mu), (q_i - \mu)^T E_2 (q_i - \mu)) \qquad (2.34)
\end{aligned}
$$

where notation $\arctan 2(Y, X) = tan^{-1}(Y/X)$ is restricted to take values in $(-\pi, \pi)$. Thus set of detectable behaviors (2.9) for map ϕ is given by (2.17), (2.31), and (2.34).

Since the co-distributions as defined in (2.6) are

$$
\Omega_g = \operatorname{span}\{d\mu, d\theta\}, \quad \Omega_s = \operatorname{span}\{ds_1, ds_2\}
$$

and the control distributions corresponding to $\bar{\Delta}_g$ and $\bar{\Delta}_s$ are given by

$$
\bar{\Delta}_g = \operatorname{span}\{X_q^{\mu}, X_q^{\theta}\} \qquad (2.35)
$$

$$
\bar{\Delta}_s = \operatorname{span}\{X_q^{s_1}, X_q^{s_2}\} \qquad (2.36)
$$

where

$$
X_q^{\mu} = \begin{pmatrix} I_2 \\ \cdot \\ \cdot \\ \cdot \\ I_2 \end{pmatrix} \qquad (2.37)
$$

$$
X_q^{\theta} = \begin{pmatrix} H_3(q_i - \mu) \\ \cdot \\ \cdot \\ \cdot \\ H_3(q_i - \mu) \end{pmatrix} \qquad (2.38)
$$

$$
X_q^{s_1} = \begin{pmatrix} H_1(q_i - \mu) \\ \cdot \\ \cdot \\ \cdot \\ H_1(q_i - \mu) \end{pmatrix} \qquad (2.39)
$$

$$
X_q^{s_2} = \begin{pmatrix} H_2(q_i - \mu) \\ \cdot \\ \cdot \\ \cdot \\ H_2(q_i - \mu) \end{pmatrix} \qquad (2.40)
$$

Thus, in accordance with (2.9), point 1 of **Problem** defined in 2.1.1 is satisfied if the behaviours are restricted to the set $\bar{\Delta_g} + \bar{\Delta_s}$ as given by (2.37) - (2.40).

In order for the control directions to be independent of each other, we need to have $\bar{\Delta_g}$ and $\bar{\Delta_s}$ orthogonal to each other so that decoupled controls can be designed for group and shape in accordance with point 3 of the **Problem** 2.1.1.

It is easy to see that the two columns of X_q^{μ} are orthogonal. X_q^{θ}, $X_q^{s_1}$ and $X_q^{s_2}$ are orthogonal to X_q^{μ} by the definition of μ (2.17). Since $H_1H_2 = 0$, $X_q^{s_1}$ and $X_q^{s_2}$ are also orthogonal. Combining $H_2H_3 = H_3 + E_3$, $E_3^T = -E_3$ with (2.30), X_q^{θ} is orthogonal to both $X_q^{s_1}$ and $X_q^{s_2}$. Thus, it can be said that $\bar{\Delta_g}$ and $\bar{\Delta_s}$ are orthogonal. Thus, point 3 of **Problem** 2.1.1 is verified. Also, since the control directions $X_q^{\mu}, X_q^{\theta}, X_q^{s_1}, X_q^{s_2}$ are chosen as the basis for $\bar{\Delta_g}$ and $\bar{\Delta_s}$ as defined in (2.35) and (2.36), each of the formation variables can be individually controlled. These individual control laws are discussed in the next section.

2.5 Individual Control Laws

In this section, control laws are defined and evaluated based on the conditions shown in previous sections. The control laws are based on the architecture shown in Figure 2.1 with some modifications. According to the figure, the control law determined by the controller C_i for each robot R_i is only dependent on its state q_i and the abstract state a which is updated by an observer. This observer does the job of collecting information (states q_i) from all the robots and updating the abstract state x according to the mapping defined by ϕ. The Abstract motion planner prescribes the desired abstract final trajectory a^d and the desired speed of convergence k_a. Thus, this architecture involves minimum communication between robots. But, there is a major drawback in this method as it does not consider the collision free environment for robots i.e movement of robots are based on their previous sate and previous abstract state and thus one robot does not know in what direction the other robot is moving and will only come to know of the other robots position after they have moved. Thus collision cannot be avoided using this architecture. The solution to this is addressed in CHAPTER 3 and CHAPTER 4.

From (2.35) and (2.36), we have

$$\dot{X} = \text{span}\{\bar{\Delta_g}, \bar{\Delta_s}\}$$
$$\dot{X} = \text{span}\{X_q^{\mu}, X_q^{\theta}, X_q^{s_1}, X_q^{s_2}\}$$

We know that from (2.11)

$$d\phi\dot{q} = \dot{x}$$

and $\dot{x} = (\dot{u}, \dot{\theta}, \dot{s_1}\dot{s_2})$. In Matrix form, it can be written as

$$\dot{x} = \begin{pmatrix} \dot{u} \\ \dot{\theta} \\ \dot{s_1} \\ \dot{s_2} \end{pmatrix}_{\text{5x1}} \qquad \dot{q} = \begin{pmatrix} \dot{q_1} \\ \dot{q_2} \\ . \\ . \\ . \\ \dot{q_N} \end{pmatrix}_{\text{Nx1}} \qquad d\phi = \begin{pmatrix} d\phi_1 \\ d\phi_2 \\ d\phi_3 \\ d\phi_4 \end{pmatrix}_{\text{5xN}} \tag{2.41}$$

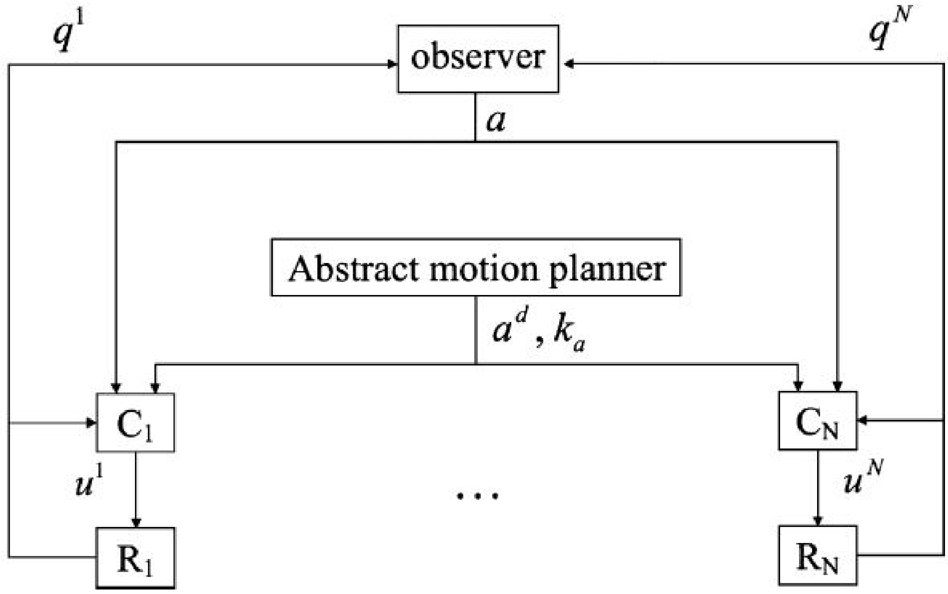


Figure 2.1: Control and Communication Architecture

Thus, substituting this in (2.11), we get

$$\begin{pmatrix} \dot{u} \\ \dot{\theta} \\ \dot{s_1} \\ \dot{s_2} \end{pmatrix} = \begin{pmatrix} d\phi_1 \\ d\phi_2 \\ d\phi_3 \\ d\phi_4 \end{pmatrix}_{\text{5xN}} \begin{pmatrix} \dot{q_1} \\ \dot{q_2} \\ . \\ . \\ . \\ \dot{q_N} \end{pmatrix}_{\text{Nx1}} \tag{2.42}$$

Note that, μ and $d\phi$ are having two rows each for x and y dimensions. From the previous values of μ (2.17), θ (2.34) and s_1, s_2 (2.31), we have

$$\begin{aligned} \mu &= \frac{1}{N}\sum_{i=1}^{N} q_i \\ d\mu &= \frac{1}{N}\sum_{i=1}^{N} I_2 \dot{q}_i \\ \dot{\mu} &= \frac{1}{N}[I_2 ... I_2]_{\text{2xN}}\ \dot{q}_{\text{Nx1}} \end{aligned} \tag{2.43}$$

and

$$\begin{aligned} s_1 &= \frac{1}{2(N-1)}\sum_{i=1}^{N}(q_i-\mu)^T H_1 (q_i-\mu) \\ ds_1 &= \frac{1}{(N-1)}\sum_{i=1}^{N}(q_i-\mu)^T H_1 \dot{q}_i \\ \dot{s_1} &= \frac{1}{(N-1)}[(q_1-\mu)^T H_1 .... (q_N-\mu)^T H_1]_{\text{2xN}}\ \dot{q}_{\text{Nx1}} \end{aligned} \tag{2.44}$$

and

$$
\begin{aligned}
s_2 &= \frac{1}{2(N-1)} \sum_{i=1}^{N} (q_i-\mu)^T H_2 (q_i-\mu) \\
ds_2 &= \frac{1}{(N-1)} \sum_{i=1}^{N} (q_i-\mu)^T H_2 \dot{q}_i \\
\dot{s_2} &= \frac{1}{(N-1)} [(q_1-\mu)^T H_2 .... (q_N-\mu)^T H_2]_{\text{2xN}}\ \dot{q}_{\text{Nx1}}
\end{aligned} \tag{2.45}
$$

and

$$
\begin{aligned}
d\theta &= \frac{1}{(N-1)(s_1-s_2)} \sum_{i=1}^{N} (q_i-\mu)^T H_3 \dot{q}_i \\
\dot{\theta} &= \frac{1}{(N-1)(s_1-s_2)} [(q_1-\mu)^T H_3 .... (q_N-\mu)^T H_3]_{\text{2xN}}\ \dot{q}_{\text{Nx1}}
\end{aligned} \tag{2.46}
$$

Combining these with (2.42) we get,

$$
d\phi = \begin{pmatrix}
\frac{I_2}{N} & \dots & \frac{I_2}{N} \\
\frac{1}{(N-1)(s_1-s_2)}(q_1-\mu)^T H_3 & \dots & \frac{1}{(N-1)(s_1-s_2)}(q_N-\mu)^T H_3 \\
\frac{1}{(N-1)}(q_1-\mu)^T H_1 & \dots & \frac{1}{(N-1)}(q_N-\mu)^T H_1 \\
\frac{1}{(N-1)}(q_1-\mu)^T H_2 & \dots & \frac{1}{(N-1)}(q_N-\mu)^T H_2
\end{pmatrix}
$$

It can be rewritten as

$$
d\phi = \frac{1}{N-1} \begin{pmatrix}
\frac{N-1}{N} I_2 & \dots & \frac{N-1}{N} I_2 \\
\frac{1}{(s_1-s_2)}(q_1-\mu)^T H_3 & \dots & \frac{1}{(s_1-s_2)}(q_N-\mu)^T H_3 \\
(q_1-\mu)^T H_1 & \dots & (q_N-\mu)^T H_1 \\
(q_1-\mu)^T H_2 & \dots & (q_N-\mu)^T H_2
\end{pmatrix} \tag{2.47}
$$

In previous section it was shown that $X_q^{\mu}, X_q^{\theta}, X_q^{s_1}, X_q^{s_2}$ are mutually orthogonal to each other. Thus, changing one does not affect the other parameter. It can also be interpreted as controlling each variable individually without affecting the other. Using (2.14) and (2.47), we can write,

$$
\begin{aligned}
\dot{q}_i &= d\phi^T (d\phi d\phi^T)^{-1} \dot{x} \\
\dot{q}_i &= \dot{\mu} X_q^{\mu} + \frac{s_1-s_2}{s_1+s_2} \dot{\theta} X_q^{\theta} + \frac{1}{4s_1} \dot{s_1} X_q^{s_1} + \frac{1}{4s_1} \dot{s_2} X_q^{s_2} \\
\dot{q}_i &= \dot{\mu} + \frac{s_1-s_2}{s_1+s_2} H_3 (q_i-\mu) \dot{\theta} + \frac{1}{4s_1} H_1 (q_i-\mu) \dot{s_1} + \frac{1}{4s_1} H_2 (q_i-\mu) \dot{s_2} \\
u_i &= \dot{\mu} + \frac{s_1-s_2}{s_1+s_2} H_3 (q_i-\mu) \dot{\theta} + \frac{1}{4s_1} H_1 (q_i-\mu) \dot{s_1} + \frac{1}{4s_1} H_2 (q_i-\mu) \dot{s_2}
\end{aligned} \tag{2.48}
$$

The above control law, fits the diagram 2.1 wherein each robot implements a controllers that depends on it's own state and the abstract dimensional state x. We are converting the given state to abstract state at each instant of time and finding the optimal control law and then obtain the equivalent control law in world frame and apply it to the robot.

IMPORTANT NOTES:

1. Case when $s_1 = 0$ and $s_2 = 0$ is not defined by the above control law. The abstract behavior should be designed such that $s_1 > 0$ and $s_2 > 0, \forall t$. This has a physical significance. When $s_1 = 0, s_2 = 0$, all the robots are on the origin of the virtual frame. This would be a degenerate case.

2. Case when $s_1 = s_2$. For this case, the derivative of θ is not defined. Physically, this would mean that the robots are equally distributed along the axes of the virtual frame and thus there would not be any information regarding the orientation of the robots in the virtual frame as in this would be circle. For circle, there are no major and minor axes. This would mean that there are infinitely many combinations of axes that can be obtained from the circle. Thus, no orientation information can be obtained when $s_1 = s_2$. Mathematically, the equations (2.31) and (2.34) are not defined for $s_1 = s_2$ case.

2.6 Abstraction

Previous chapter gave the control law (2.48) which will be implemented by the controller C_i for each robot. For every instant of time, the observer collects states q_i from all the robots and updates the abstract state x for that instant of time according to the equations (2.17), (2.31), (2.34) and sends it to all the controllers. For the next instant of time, each controller knows where other robots are through the abstract state x as updated by the observer, and thus based on this information and its current state q_i, a control signal is generated in the virtual frame perspective and applied to each robot and is updated to the observer as provided by the equation (2.48). This process repeats until the desired abstract state is reached. As mentioned earlier, this does not take into account the safety requirement of robots i.e. inter-robot collision mechanism does not exist in the architecture. This will be addressed in later part of the report.

If the goal is to move the robots from their arbitrary initial positions $q_i(0)$ to final rest positions of desired mean μ^d, orientation θ^d, and shape s_1^d and s_2^d, an appropriate choice of the control vector field $\dot{x} = [\dot{\mu}, \dot{\theta}, \dot{s_1}, \dot{s_2}]$ on the abstract manifold M is

$$\begin{aligned}
\dot{\mu} &= K_\mu(\mu^d - \mu) \\
\dot{\theta} &= k_\theta(\theta^d - \theta) \\
\dot{s_1} &= k_{s_1}(s_1^d - s_1) \\
\dot{s_2} &= k_{s_2}(s_2^d - s_2)
\end{aligned} \tag{2.49}$$

where $K_\mu \in \mathbb{R}^{2x2}$ is a positive definite matrix and $k_\theta, k_{s_{1,2}} > 0$. In general task of each robot is to follow a desired trajectory $x^d(t) = [\mu^d(t), \theta^d(t), s_1^d(t), s_2^d(t)]$ on M. Thus control vector field on M can be of the form

$$\begin{aligned}
\dot{\mu} &= K_\mu(\mu^d(t) - \mu(t)) + \dot{\mu}^d(t) \\
\dot{\theta} &= k_\theta(\theta^d(t) - \theta(t)) + \dot{\theta}^d(t) \\
\dot{s_1} &= k_{s_1}(s_1^d(t) - s_1(t)) + \dot{s}_1^d(t) \\
\dot{s_2} &= k_{s_2}(s_2^d(t) - s_2(t)) + \dot{s}_2^d(t)
\end{aligned} \tag{2.50}$$

Later in Proposition 2 2.6 we show (using Lyapunov stability criteria) that the control law is designed in such a way that the system stabilizes once it converges to the desired state. This is also proved in the simulations. Thus $\dot{\mu}^d(t) = 0, \dot{\theta}^d(t) = 0, \dot{s}_1^d(t) = 0, \dot{s}_2^d(t) = 0$. It is important to note that the (2.50) only guarantees the desired behavior in the abstract manifold M. If the imposed trajectory $x^d(t)$ is bounded at all times, $x(t)$ is bounded. But, it is also required that the system defined by q in the configuration space Q is also bounded. Thus the following proposition is proposed.

Proposition 1: If abstract state x is bounded, then the system defined by $q_i, i = 1,2,....,N$ in the configuration space Q, is also bounded.

Proof: x is bounded when μ, s_1, s_2 are bounded. Thus, we need to show that q_i is bounded if μ, s_1, s_2 are bounded. Let us consider that μ, s_1, s_2 are bounded. Therefore,

$$\|\mu - \mu^d\| \leq M_\mu \tag{2.51}$$

$$|s_1 - s_1^d| \leq M_{s1} \tag{2.52}$$

$$|s_2 - s_2^d| \leq M_{s2} \tag{2.53}$$

From (2.31), we can obtain the following:

$$s_1 + s_2 = \frac{1}{N-1} \sum_{i=1}^{N} (q_i - \mu)^T (q_i - \mu)$$

$$(N-1)(s_1 + s_2) = \sum_{i=1}^{N} (q_i - \mu)^T (q_i - \mu) \tag{2.54}$$

Thus, using (2.52) and (2.53) and noting that LHS is the norm of $(q_i - \mu)$, we can write:

$$\begin{aligned} \|(q_i - \mu)\| &\leq \sqrt{N(s_1 + s_2)} \\ &\leq \sqrt{(N-1)(M_{s1} + M_{s2} + s_1^d + s_2^d)} \end{aligned} \tag{2.55}$$

Now, using (2.51), we can write

$$\begin{aligned} \|q_i - \mu^d\| &= \|q_i - \mu + \mu - \mu^d\| \\ &\leq \|q_i - \mu\| + \|\mu - \mu^d\| \\ \|q_i - \mu^d\| &\leq \sqrt{(N-1)(M_{s1} + M_{s2} + s_1^d + s_2^d)} + M_u \end{aligned} \tag{2.56}$$

This shows that if μ, s_1, s_2 are bounded, q_i is also bounded. From this proof it can be concluded that the system defined by q is bounded given that the abstract state x is bounded. This showed the boundness of configuration space also, but we also need to ensure that the system is stable once it reaches the desired state i.e. is the desired state an equilibrium state for the abstract state x? Also, how does the system converge to the desired state? The following proof demonstrates the answers to these queries.

Proposition 2: For any $u^d, \theta^d, s_1^d, s_d^2$, the closed loop system globally asymptotically converges to the equilibrium manifold $\mu = \mu^d, \theta = \theta^d, s_1 = s_1^d, s_2 = s_d^2$.

Proof: We need to show that $\dot{\mu}^d(t) = 0, \dot{\theta}^d(t) = 0, \dot{s}_1^d(t) = 0, \dot{s}_2^d(t) = 0$, i.e. the system is stable once it reaches the desired state and thus conclude that the desired state is an equilibrium state. When each robot is in equilibrium, i.e. $(\dot{q}_i) = 0, i = 1, ..., N$, the abstract state is also in equilibrium i.e. $\dot{x} = 0$. This can be easily seen from equations, where all the state variables $d\theta, d\mu, ds_1, ds_2$ or equivalently written as $\dot{\theta}, \dot{\mu}, \dot{s_1}, \dot{s_2}$ are dependent on $\dot{q}_i$

$$
\begin{aligned}
\mu &= \frac{1}{N}\sum_{i=1}^{N} q_i \\
d\theta &= \frac{1}{(N-1)(s_1 - s_2)}\sum_{i=1}^{N}(q_i - \mu)^T H_3 \dot{q}_i \\
ds_1 &= \frac{1}{(N-1)}\sum_{i=1}^{N}(q_i - \mu)^T H_1 \dot{q}_i \\
ds_2 &= \frac{1}{(N-1)}\sum_{i=1}^{N}(q_i - \mu)^T H_2 \dot{q}_i \\
u_i &= \dot{\mu} + \frac{s_1 - s_2}{s_1 + s_2} H_3 (q_i - \mu)\dot{\theta} + \frac{1}{4s_1} H_1 (q_i - \mu)\dot{s_1} + \frac{1}{4s_1} H_2 (q_i - \mu)\dot{s_2}
\end{aligned} \tag{2.57}
$$

Now to prove the asymptotic convergence, a Lyapunov function is considered as follows:

$$
V(q) = \frac{1}{2}\|\mu^d - \mu\|^2 + \frac{1}{2}(\theta^d - \theta)^2 + \frac{1}{2}(s_1^d - s_1)^2 + \frac{1}{2}(s_2^d - s_2)^2 \tag{2.58}
$$

Next, it's derivative along the vector field on Q is considered as:

$$
\dot{V}(q) = -K_\mu \|\mu^d - \mu\|^2 - k_\theta(\theta^d - \theta)^2 - k_{s_1}(s_1^d - s_1)^2 - k_{s_2}(s_2^d - s_2)^2 \tag{2.59}
$$

where, $K_\mu, k_\theta, k_{s_1}, k_{s_2} > 0$ and $K_\mu > 0$ is meant in the positive definite sense. Thus, $\dot{V}(q) \leq 0, \forall q \in \mathbb{R}^{2N}$. Also, $V(q)$ is a Lyapunov function candidate because its derivative converges to zero for all q_i, and $\dot{V} = 0$ only when the states reach their desired state, i.e. $\mu = \mu^d, \theta = \theta^d, s_1 = s_1^d, s_2 = s_d^2$, which is an invariant set for the closed loop system i.e desired position is not changed once set (it is not dynamic). According to the global invariant set theorem from LaSalle, the set must converge to the largest invariant set, i.e. $V(q) \to \infty$ as $\|q\| \to \infty$.

Proposition 3: LaSalle's theorem applies to the invariant set $\mu = \mu^d, \theta = \theta^d, s_1 = s_1^d, s_2 = s_d^2$ and thus $V(q) \to \infty$ as $\|q\| \to \infty$.

Proof: Proof by method of contradiction, i.e. we will show that if an assumption that as $\|q\| \to \infty$ is made, and if $V(q)$ does not tend to ∞, the assumption that $\|q\| \to \infty$ is contradicted. Thus, as $\|q\| \to \infty$, $V(q) \to \infty$.

Suppose, $\|q\| \to \infty$ and $\exists$ some $L > 0$ such that $V(q) < L$, i.e. $V(q) \not\to \infty$. This would imply that the states are bounded and is given by

$$
\begin{aligned}
\|\mu - \mu^d\| &\leq \sqrt{2L} \\
|s_1 - s_1^d| &\leq \sqrt{2L} \\
|s_2 - s_2^d| &\leq \sqrt{2L}
\end{aligned} \tag{2.60}
$$

Similar to Proposition 2 in 2.6, it can be shown that

$$\|q_i - \mu^d\| \leq \sqrt{(N-1)(2\sqrt{2L} + s_1^d + s_2^d)} + \sqrt{2L} \tag{2.61}$$

This means that all q_i are bounded $\forall i = 1, \ldots.., N$. But, it was assumed in the beginning that $\|q\| \to \infty$. This would imply that for at least one $i = 1, \ldots.., N$, $\|q_i\| \to \infty$. This is a contradiction. Thus, $V(q) \to \infty$ as $\|q\| \to \infty$.
This proves that the choice of states in (2.49) is stable once is reaches the desired state and it reaches the desired state by converging asymptotically.

Chapter 3

MOVING FRAME

This section deals with the application of the concept demonstrated earlier to a moving frame and obtaining control laws from moving frame B back to the world frame or inertial frame W.

Let's define a moving frame B as shown in Figure 3.1 whose origin is at the centroid of the world frame, by requiring the orientation to be such that the coordinates of the robots in this frame,

$$p_i = [x_i, y_i] = R^T(q_i - \mu) \tag{3.1}$$

satisfy

$$\sum_{i=1}^{N} x_i y_i = 0 \tag{3.2}$$

where the parameterization is defined by:

$$R = \begin{pmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{pmatrix} \in \mathbb{R}^2. \tag{3.3}$$

Hence, the distribution of the robots in this local frame is approximated by the inertia tensor(assuming uniform unit mass) or by matrix of second moments or co-variance matrix:

$$I = p_i p_i^T = \begin{pmatrix} I_{11} & 0 \\ 0 & I_{22} \end{pmatrix} \tag{3.4}$$

Since we are dealing with shapes in 2-D, $s = (s1, s2)$ is taken to be proportional to the diagonal elements:

$$\begin{aligned} s_1 = kI_{11} &= \frac{1}{N-1} \sum_{i=1}^{N} x_i^2 \\ s_2 = kI_{22} &= \frac{1}{N-1} \sum_{i=1}^{N} y_i^2 \end{aligned} \tag{3.5}$$

where $k > 0$. It is easy to observe that when k is chosen as $\frac{1}{N-1}$, $s1$ and $s2$ are geometrically equivalent to semi-major and semi-minor axes of an ellipse which encompasses set of points that satisfy normal distribution. In this case, the set of points are groups of robots which are on a plane.

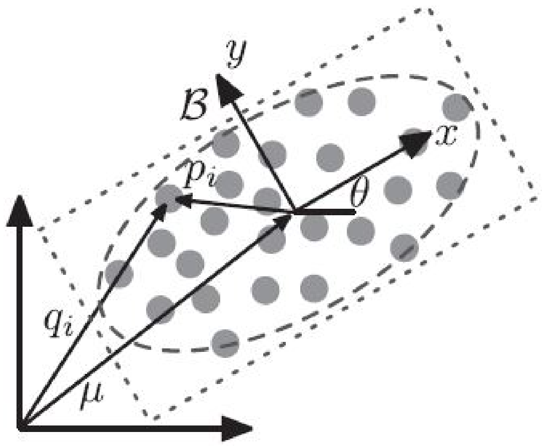


Figure 3.1: Frame B is fixed to the robots and moves with them and is oriented with respect to world frame W by θ.

Since $R \in SO(2)$ is one dimensional (1-D) i.e. 1 Degree of Freedom (DOF); θ is sufficient to describe R, the dimension of the abstract manifold, $M = (g,s) = (R,\mu,s), \mu \in \mathbb{R}^2, s \in \mathbb{R}^2$ is n=5, independent of the number of robots N. Here, g is the position and orientation of the moving frame B, given by:

$$g = \begin{pmatrix} cos(\theta) & -sin(\theta) & \mu_1 \\ sin(\theta) & cos(\theta) & \mu_2 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.6}$$

where $\mu = (\mu_1, \mu_2)$ are the components of the centroid of the inertial frame and shape $s = (s_1, s_2)$.

3.1 Dynamics of the Moving Frame

Now that a moving frame is defined, we would want control laws for robots in this frame. At any point $x = (g,s) \in M$ in the abstract space, the derivative in the moving frame is given by

$$\dot{x} = \begin{pmatrix} \dot{g} \\ \dot{s} \end{pmatrix} = \begin{pmatrix} g & 0 \\ 0 & I_2 \end{pmatrix} \begin{pmatrix} \xi \\ \sigma \end{pmatrix} \tag{3.7}$$

Here, $\dot{x} = (\dot{g}, \dot{s})$ is the time derivative of the abstract space in the inertial frame and $\zeta = (\xi, \sigma)$ is the time derivative in the moving frame B and

$$\Gamma = \begin{pmatrix} g & 0 \\ 0 & I_2 \end{pmatrix} \tag{3.8}$$

is a non-singular 5x5 transformation matrix. From this transformation matrix we can now relate the control law in the moving frame with the control law in the inertial frame. If v_i is the robot velocity in frame B, the inertial frame velocity is given by $u_i = Rv_i$. From this we get the relation for v given by $v_i = R^T u_i$. From (2.11) and (2.47), we have,

$$
\begin{aligned}
\dot{x} &= d\phi \dot{q} \\
\begin{pmatrix} g & 0 \\ 0 & I_2 \end{pmatrix} \begin{pmatrix} \xi \\ \sigma \end{pmatrix} &= d\phi u \\
\begin{pmatrix} g & 0 \\ 0 & I_2 \end{pmatrix} \begin{pmatrix} \xi \\ \sigma \end{pmatrix} &= k \begin{pmatrix} \frac{N-1}{N} I_2 & \dots & \frac{N-1}{N} I_2 \\ \frac{1}{(s_1-s_2)} (q_1-\mu)^T H_3 & \dots & \frac{1}{(s_1-s_2)} (q_N-\mu)^T H_3 \\ (q_1-\mu)^T H_1 & \dots & (q_N-\mu)^T H_1 \\ (q_1-\mu)^T H_2 & \dots & (q_N-\mu)^T H_2 \end{pmatrix} u \\
\begin{pmatrix} g & 0 \\ 0 & I_2 \end{pmatrix} \begin{pmatrix} \xi \\ \sigma \end{pmatrix} &= k \begin{pmatrix} \frac{N-1}{N} I_2 & \dots & \frac{N-1}{N} I_2 \\ \frac{1}{(s_1-s_2)} (q_1-\mu)^T R^2 E_1 & \dots & \frac{1}{(s_1-s_2)} (q_N-\mu)^T R^2 E_1 \\ (q_1-\mu)^T (I_2+R^2E_2) & \dots & (q_N-\mu)^T (I_2+R^2E_2) \\ (q_1-\mu)^T (I_2-R^2E_2) & \dots & (q_N-\mu)^T (I_2-R^2E_2) \end{pmatrix} u \\
\begin{pmatrix} g & 0 \\ 0 & I_2 \end{pmatrix} \begin{pmatrix} \xi \\ \sigma \end{pmatrix} &= k \begin{pmatrix} \frac{N-1}{N} I_2 & \dots & \frac{N-1}{N} I_2 \\ \frac{1}{(s_1-s_2)} (q_1-\mu)^T R^2 E_1 & \dots & \frac{1}{(s_1-s_2)} (q_N-\mu)^T R^2 E_1 \\ (q_1-\mu)^T + (q_1-\mu)^T R^2 E_2) & \dots & (q_N-\mu)^T + (q_N-\mu)^T R^2 E_2) \\ (q_1-\mu)^T - (q_1-\mu)^T R^2 E_2) & \dots & (q_N-\mu)^T - (q_N-\mu)^T R^2 E_2) \end{pmatrix} u \\
\begin{pmatrix} g & 0 \\ 0 & I_2 \end{pmatrix} \begin{pmatrix} \xi \\ \sigma \end{pmatrix} &= k \begin{pmatrix} \frac{1}{kN} I_2 & \dots & \frac{1}{kN} I_2 \\ \frac{1}{(s_1-s_2)} p_1^T E_1 R^T & \dots & \frac{1}{(s_1-s_2)} p_N^T E_1 R^T \\ (q_1-\mu)^T + p_1^T E_2 R^T & \dots & (q_N-\mu)^T + p_N^T E_2 R^T \\ (q_1-\mu)^T - p_1^T E_2 R^T & \dots & (q_N-\mu)^T - p_N^T E_2 R^T \end{pmatrix} Rv \\
\begin{pmatrix} \xi \\ \sigma \end{pmatrix} &= \begin{pmatrix} g & 0 \\ 0 & I_2 \end{pmatrix}^{-1} k \begin{pmatrix} \frac{1}{kN} I_2 R & \dots & \frac{1}{kN} I_2 R \\ \frac{1}{(s_1-s_2)} p_1^T E_1 R^T R & \dots & \frac{1}{(s_1-s_2)} p_N^T E_1 R^T R \\ (q_1-\mu)^T + p_1^T E_2 R^T R & \dots & (q_N-\mu)^T + p_N^T E_2 R^T R \\ (q_1-\mu)^T - p_1^T E_2 R^T R & \dots & (q_N-\mu)^T - p_N^T E_2 R^T R \end{pmatrix} v
\end{aligned}
$$

$$
\begin{aligned}
\begin{pmatrix} \xi \\ \sigma \end{pmatrix} &= k \begin{pmatrix} \frac{1}{kN} I_2 & \dots & \frac{1}{kN} I_2 \\ \frac{1}{(s_1-s_2)} p_1^T E_1 & \dots & \frac{1}{(s_1-s_2)} p_N^T E_1 \\ p_1^T + p_1^T E_2 & \dots & p_N^T + p_N^T E_2 \\ p_1^T - p_1^T E_2 & \dots & p_N^T - p_N^T E_2 \end{pmatrix} v \\
\begin{pmatrix} \xi \\ \sigma \end{pmatrix} &= k \begin{pmatrix} \frac{1}{kN} I_2 & \dots & \frac{1}{kN} I_2 \\ \frac{1}{(s_1-s_2)} p_1^T E_1 & \dots & \frac{1}{(s_1-s_2)} p_N^T E_1 \\ p_1^T (I_2+E_2) & \dots & p_N^T (I_2+E_2) \\ p_1^T (I_2-E_2) & \dots & p_N^T (I_2-E_2) \end{pmatrix} \begin{pmatrix} v_1 \\ . \\ . \\ . \\ v_N \end{pmatrix} \qquad (3.9)
\end{aligned}
$$

The minimum-energy solution to the above equation is analogous to (2.48) and can be written as

$$
v^* = d\phi^T (d\phi d\phi^T)^{-1} \zeta \qquad (3.10)
$$

and this can be simplified as shown in (2.48) as,

$$v_i^* = \dot{\xi} + \frac{s_1 - s_2}{s_1 + s_2} E_1 p_i \xi_3 + \frac{1}{4s_1}(I_2 + E_2) p_i \sigma_1 + \frac{1}{4s_2} p_i \sigma_2$$

$$v_i^* = \begin{pmatrix} \dot{\xi}_1 \\ \dot{\xi}_2 \end{pmatrix} + \frac{s_1 - s_2}{s_1 + s_2} E_1 p_i \xi_3 + \frac{1}{4s_1}(I_2 + E_2) p_i \sigma_1 + \frac{1}{4s_2}(I_2 - E_2) p_i \sigma_2 \tag{3.11}$$

We now have the control law that defines the movement of robots in local frame B confining them to a particular shape as defined by σ_1 and σ_2. But, there is a drawback to this method of choosing control law. It does not provide any rules for collision avoidance, i.e. there are no conditions on the robots to avoid collision while achieving the desired orientation and position while satisfying the constraint of an ellipse. Thus, we need to deal with collision avoidance for safe movement of robots.

3.2 Collision Avoidance

For collision avoidance, a safe separation distance is considered between each robots. The safe separation distance is a summation of the diameter of the robot plus a specified safety region to avoid collision between the robots. Each bot gets the information of all its neighbouring bots. In order to satisfy the condition of collision the following inequality constraint has to hold good.

$$\varepsilon = 2\rho + \varepsilon_s \tag{3.12}$$

where ρ is the radius of each robot and ε_s is the safety separation distance. Here, radius of the robot does not necessarily mean that the robot is circular, it's just an outer boundary covering each robot.

The inequality constraint will be considered in the optimization problem only when the magnitude of the distance is lesser than equal to the safe separation distance between each of the neighbours. On satisfying the inequality constraint, the robots do not converge and collide against each other and this ensures collision free mechanism. The separation distance between any two robots using their reference points is given as:

$$\delta_{ij} = \|p_i - p_j\| \tag{3.13}$$

If a neighbourhood $\mathcal{N}_i$ is defined as the set of all robots that are sensed by the robot i, to ensure collision free movement of robots, we need to have

$$(p_i - p_j).(v_i - v_j)^T \geq 0 \quad \forall j \in \mathcal{N}_i \qquad when \quad \delta_{ij} \leq \varepsilon \tag{3.14}$$

NOTE: We used the condition mentioned above that is different from what the author has given:

$$(p_i - p_j).(v_i - v_j) \geq 0 \tag{3.15}$$

As per our analysis the simulations work for (3.14). (3.15) does not match in the proof of Proposition 6: 4.2.1.

3.3 Asymptotic Convergence to a Desired State

Let x^{des} be the desired abstract state. The author has considered the desired abstract state to be time invariant. This implies that there is no change in the desired state once the robots are fed the desired state as an input i.e. there is no dynamic change in the desired state. Once the robots reach their desired state, another desired state can be given as an input but this input cannot be given before the robots achieve their current desired state. There are several ways to achieve the desired state, but the easiest and safest option when multi-robot system is considered is to converge the state error $\tilde{x} = (x^{\text{des}} - x)$ exponentially to zero. Thus, the equivalent state can be written as

$$\dot{x} = K\tilde{x} \tag{3.16}$$

where K is any positive definite matrix. When $x^{\text{des}} = 0$, we are at equilibrium, thus $\tilde{x} = 0$. It was shown earlier that $\dot{x} = \Gamma\zeta$. Thus, the above expression can be written as:

$$\zeta = \Gamma^{-1}Kx^{\text{des}} \tag{3.17}$$

When this condition is plugged in (3.11), we obtain robot velocities that guarantee globally asymptotic convergence to any abstract state.

We have following conditions that the robot velocities/control inputs must satisfy

1. the minimum-energy solution condition; done in (3.11)
2. state to converge monotonically to an abstract state
3. robots to avoid inter-robot collisions given by (3.14)

Chapter 4

CONTROL WITH COLLISION AVOIDANCE

4.1 Monotonic Convergence

The author has not considered the exact exponential convergence to an abstract state. This is reasonable as the minimum-energy solution obtained may not satisfy the safety constraints always and it doesn't the collision avoidance will not work. Thus, he finds a solution closest to minimum-energy solution that satisfies safety constraints.

The error in the abstract case decreases monotonically, thus we have:

$$\tilde{x}^T K \dot{x} \geq 0 \tag{4.1}$$

Substituting for $\dot{x}$, we get

$$\tilde{x}^T K\Gamma \begin{pmatrix} I_2 & \dots & I_2 \\ \frac{1}{(s_1-s_2)} p_1^T E_1 & \dots & \frac{1}{(s_1-s_2)} p_N^T E_1 \\ p_1^T(I_2+E_2) & \dots & p_N^T(I_2+E_2) \\ p_1^T(I_2-E_2) & \dots & p_N^T(I_2-E_2) \end{pmatrix} \begin{pmatrix} v_1 \\ . \\ . \\ . \\ v_N \end{pmatrix} \geq 0 \tag{4.2}$$

Thus, each robot should satisfy the condition

$$\tilde{x}^T K\Gamma \begin{pmatrix} I_2 \\ \frac{1}{(s_1-s_2)} p_i^T E_1 \\ p_i^T(I_2+E_2) \\ p_i^T(I_2-E_2) \end{pmatrix} v_i \geq 0 \tag{4.3}$$

Thus, if all the robots satisfy control law as given by (3.11), the error in the abstract state will monotonically decrease. It is interesting to note that, minimum-energy control law satisfies the above condition. Thus, while finding the solution closest to minimum-energy control law solution, we ensure that it still monotonically decreases the error in the abstract state.

Proposition 4: The minimum-energy control law (3.11) with ζ given by (3.17) satisfies the monotonic convergence condition (4.3).

Proof. We define g_i and m_i such that

$$
\begin{aligned}
g_i &= \left[I_2, \frac{1}{s_1 - s_2} p_i^T E_1, p_i^T (I_2 + E_2), p_i^T (I_2 - E_2)\right]^T \\
m_i &= \left[I_2, \frac{s_1 - s_2}{s_1 + s_2} E_1 p_i, \frac{1}{4s_1}(I_2 + E_2) p_i, \frac{1}{4s_2}(I_2 - E_2) p_i\right]
\end{aligned}
\tag{4.4}
$$

Substituting (3.11) and (4.4) into the left hand side of (4.3) gives:

$$
\tilde{x}^T K \Gamma \begin{pmatrix} I_2 \\ \frac{1}{(s_1 - s_2)} p_i^T E_1 \\ p_i^T (I_2 + E_2) \\ p_i^T (I_2 - E_2) \end{pmatrix} v_i \geq 0
$$

$$
\tilde{x}^T K \Gamma \begin{pmatrix} I_2 \\ \frac{1}{(s_1 - s_2)} p_i^T E_1 \\ p_i^T (I_2 + E_2) \\ p_i^T (I_2 - E_2) \end{pmatrix}_{4\text{x}2} \left[\begin{pmatrix} \dot{\xi}_1 \\ \dot{\xi}_2 \end{pmatrix} + \frac{s_1 - s_2}{s_1 + s_2} E_1 p_i \xi_3 + \frac{1}{4s_1}(I_2 + E_2) p_i \sigma_1 + \frac{1}{4s_2}(I_2 - E_2) p_i \sigma_2 \right]_{2\text{x}1} \geq 0
$$

$$
\begin{aligned}
\tilde{x}^T K \Gamma g_i m_i \begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \\ \sigma_1 \\ \sigma_2 \end{pmatrix} &\geq 0 \\
\tilde{x}^T K \Gamma g_i m_i \begin{pmatrix} \xi \\ \sigma \end{pmatrix} &\geq 0 \\
\tilde{x}^T K \Gamma g_i m_i \zeta &\geq 0 \\
\tilde{x}^T K \Gamma [g_i \;\; m_i] \Gamma^{-1} K \tilde{x} &\geq 0
\end{aligned}
\tag{4.5}
$$

where

$$[g_i \;\; m_i] = \left[I_2, \frac{1}{s_1-s_2} p_i^T E_1, p_i^T (I_2+E_2), p_i^T (I_2-E_2)\right]^T \left[I_2, \frac{s_1-s_2}{s_1+s_2} E_1 p_i, \frac{1}{4s_1}(I_2+E_2)p_i, \frac{1}{4s_2}(I_2-E_2)p_i\right]$$

$$\begin{pmatrix} I_2 \\ \frac{1}{s_1-s_2} p_i^T E_1 \\ p_i^T (I_2+E_2) \\ p_i^T (I_2-E_2) \end{pmatrix} \begin{pmatrix} I_2 & \frac{s_1-s_2}{s_1+s_2} E_1 p_i & \frac{1}{4s_1}(I_2+E_2)p_i & \frac{1}{4s_2}(I_2-E_2)p_i \end{pmatrix}$$

$$\begin{pmatrix} I_2 & \frac{s_1-s_2}{s_1+s_2} E_1 p_i & \frac{1}{4s_1}(I_2+E_2)p_i & \frac{1}{4s_2}(I_2-E_2)p_i \\ \frac{1}{s_1-s_2} p_i^T E_1 & \frac{1}{s_1+s_2} p_i^T E_1^2 p_i & \frac{1}{4s_1(s_1-s_2)} p_i^T E_1 (I_2+E_2) p_i & \frac{1}{4s_2(s_1-s_2)} p_i^T E_1 (I_2-E_2) p_i \\ p_i^T (I_2+E_2) & \frac{s_1-s_2}{s_1+s_2} p_i^T (I_2+E_2) E_1 p_i & \frac{1}{4s_1} p_i^T (I_2+E_2)^2 p_i & \frac{1}{4s_2} p_i^T (I_2+E_2)(I_2-E_2) p_i \\ p_i^T (I_2-E_2) & \frac{s_1-s_2}{s_1+s_2} p_i^T (I_2-E_2) E_1 p_i & \frac{1}{4s_1} p_i^T (I_2-E_2)(I_2+E_2) p_i & \frac{1}{4s_2} p_i^T (I_2-E_2)^2 p_i \end{pmatrix}$$

$$\begin{pmatrix} I_2 & \frac{s_1-s_2}{s_1+s_2} E_1 p_i & \frac{1}{4s_1}(I_2+E_2)p_i & \frac{1}{4s_2}(I_2-E_2)p_i \\ \frac{1}{s_1-s_2} p_i^T E_1 & \frac{1}{s_1+s_2} p_i^T \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} p_i & \frac{1}{4s_1(s_1-s_2)} p_i^T \begin{pmatrix} 0 & 0 \\ 2 & 0 \end{pmatrix} p_i & \frac{1}{4s_2(s_1-s_2)} p_i^T \begin{pmatrix} 0 & 2 \\ 0 & 0 \end{pmatrix} p_i \\ p_i^T (I_2+E_2) & \frac{s_1-s_2}{s_1+s_2} p_i^T \begin{pmatrix} 0 & 2 \\ 0 & 0 \end{pmatrix} p_i & \frac{1}{4s_1} p_i^T \begin{pmatrix} 4 & 0 \\ 0 & 0 \end{pmatrix} p_i & 0 \\ p_i^T (I_2-E_2) & \frac{s_1-s_2}{s_1+s_2} p_i^T \begin{pmatrix} 0 & 0 \\ 2 & 0 \end{pmatrix} p_i & 0 & \frac{1}{4s_2} p_i^T \begin{pmatrix} 0 & 0 \\ 0 & 4 \end{pmatrix} p_i \end{pmatrix} \tag{4.6}$$

The 5x5 matrix $[g_i \;\; m_i]$, although asymmetric, is positive semi-definite with two non-zero eigenvalues shown as follows:

$$\lambda_1 = 1 + \frac{\|p_i\|^2}{s_1+s_2} \quad \text{and} \quad \lambda_2 = 1 + \frac{p_{i,x}^2}{s_1} + \frac{p_{i,y}^2}{s_2} \tag{4.7}$$

K is chosen to be any positive definite matrix and from above we have $[g_i \;\; m_i]$ to be positive semi-definite and thus (4.5) is satisfied for positive semi-definite condition. Thus, minimum-energy control law given by (3.11) satisfies the monotonic convergence inequality (4.3).

4.2 Safe Minimum-Energy Control Law

As discussed earlier, the author has considered a solution that is closest to the solution obtained from minimum-energy control law but still satisfies the monotonic convergence and safety criteria . It is done in the following way:

Proposition 5: Equation (3.11) is a decentralized control law that selects a unique control input that has the smallest energy instantaneously while satisfying the monotonic convergence inequality and the safety constraints:

$$v_i = \arg\min_{\hat{v}_i} \|v_i^* - \hat{v}_i\|^2 \tag{4.8}$$

The above equation is subjected to conditions mentioned in equations (3.14) and (4.3).

Proof: The constraints mentioned above provides the safety and monotonic convergence condition. The function being minimized is a slight variation from the actual minimum-energy control input. The inequality constraints (3.14) and (4.3) are linear in v_i and the function being minimized is a positive definite quadratic function of v_i, the equation (4.8) is a convex, quadratic problem with a unique solution. We need to remember that each robot relies on it's own state and knowledge of error (obtained from observer), thus the control law is decentralized.

4.2.1 Convergence Properties

For showing convergence, a Lyapunov function is considered as follows:

$$V(q) = \frac{1}{2}\tilde{x}^T \tilde{x} \tag{4.9}$$

A function is a Lyapunov Function Candidate when

1. it is positive definite i.e. $V(q(t)) > 0, \forall t \neq 0$
2. $V(0) = 0$
3. and has continuous first partial derivatives in a neighborhood of the origin in $\mathbb{R}^n$ and
4. $V(q(t))$ is decreasing for increasing *time* i.e. $\dot{V}(q) < 0, \forall q(t), t > 0$

The main idea of Lyapunov stability theory is that, if $V(q)$ is decreasing for increasing time, and since V acts like a norm, the trajectory of solution of (3.16) must be converging towards the origin. That would mean $\tilde{x}$ has to converge towards the origin i.e. zero. Thus (4.9) satisfies the conditions for a function to be considered as a Lyapunov function.

Since the solution of (4.8) must satisfy the inequality (4.3), it can be inferred that $\tilde{x}^T K\dot{x} \geq 0$. If K, a positive definite matrix is chosen to be a diagonal with positive entries then it implies that $\tilde{x}^T \dot{x} \geq 0$. This is equivalent to

$$\dot{V}(q) = -\tilde{x}^T \dot{x} \leq 0 \tag{4.10}$$

Previously it was showed that q is bounded given x is bounded and that $V(q) \to \infty$ as $\|q\| \to \infty$. Also, it was shown that $V(q)$ is asymptotically stable in Proposition 2.6 and Proposition 2.6. Hence, from LaSalle's principle, abstract state will converge to the largest invariant set given by $\tilde{x}^T \dot{x} = 0$. From the equation $d\phi\dot{q} = \dot{x}$ it can inferred that the abstract state x goes to equilibrium only when input control law is zero i.e. $v = 0$. Thus, to have $v = 0$ as the only solution, the invariant set is characterized by the set of conditions that lead to the system of inequalities given by (3.14) and (4.3).

Proposition 6: For any desired change in the abstract state $\tilde{x}$, subject to the condition $\tilde{x}_4 > 0$, $\tilde{x}_5 > 0$ there is a non-zero solution to the inequalities (3.14) and (4.3).

Subject to the condition when size of the formation is increasing, it is proved that for any error abstract state $\tilde{x}$, there exists a non-trivial solution of the control law that satisfies the inequality constraints.

Proof: The solution from the minimum-energy control law is given by:

$$v_i^* = \begin{pmatrix} \dot{\xi}_1 \\ \dot{\xi}_2 \end{pmatrix} + \frac{s_1 - s_2}{s_1 + s_2} E_1 p_i \xi_3 + \frac{1}{4s_1}(I_2 + E_2) p_i \sigma_1 + \frac{1}{4s_2}(I_2 - E_2) p_i \sigma_2$$
$$v_i^* = \begin{pmatrix} \dot{\xi}_1 \\ \dot{\xi}_2 \end{pmatrix} + \frac{s_1 - s_2}{s_1 + s_2} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} \xi_3 + \frac{1}{4s_1}[\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}] \begin{pmatrix} x_i \\ y_i \end{pmatrix} \sigma_1 + \frac{1}{4s_2}[\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}] \begin{pmatrix} x_i \\ y_i \end{pmatrix}$$
$$v_i^* = \begin{pmatrix} \dot{\xi}_1 \\ \dot{\xi}_2 \end{pmatrix} + \frac{s_1 - s_2}{s_1 + s_2} \begin{pmatrix} y_i \\ x_i \end{pmatrix} \xi_3 + \frac{1}{4s_1} \begin{pmatrix} 2 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} \sigma_1 + \frac{1}{4s_2} \begin{pmatrix} 0 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} \sigma_2$$
$$v_i^* = \begin{pmatrix} \dot{\xi}_1 \\ \dot{\xi}_2 \end{pmatrix} + \frac{s_1 - s_2}{s_1 + s_2} \begin{pmatrix} y_i \\ x_i \end{pmatrix} \xi_3 + \frac{1}{4s_1} \begin{pmatrix} 2x_i \\ 0 \end{pmatrix} \sigma_1 + \frac{1}{4s_2} \begin{pmatrix} 0 \\ 2y_i \end{pmatrix} \sigma_2$$
$$v_i^* = \begin{pmatrix} \xi_1 + \frac{s_1 - s_2}{s_1 + s_2} y_i \xi_3 + \frac{x_i}{2s_1} \sigma_1 \\ \xi_2 + \frac{s_1 - s_2}{s_1 + s_2} x_i \xi_3 + \frac{y_i}{2s_2} \sigma_2 \end{pmatrix} \tag{4.11}$$

Now, it is proved that the above control law satisfies the inter-robot collision constraints for every pair of robots (i, j) and the constraint can be written as:

$$\begin{pmatrix} (x_i - x_j) & (y_i - y_j) \end{pmatrix} \begin{pmatrix} v_{i,x}^* - v_{j,x}^* \\ v_{i,y}^* - v_{j,y}^* \end{pmatrix} \geq 0 \tag{4.12}$$
$$\begin{pmatrix} (x_i - x_j) & (y_i - y_j) \end{pmatrix} (\begin{pmatrix} \xi_1 + \frac{s_1 - s_2}{s_1 + s_2} y_i \xi_3 + \frac{x_i}{2s_1} \sigma_1 \\ \xi_2 + \frac{s_1 - s_2}{s_1 + s_2} x_i \xi_3 + \frac{y_i}{2s_2} \sigma_2 \end{pmatrix} - \begin{pmatrix} \xi_1 + \frac{s_1 - s_2}{s_1 + s_2} y_j \xi_3 + \frac{x_j}{2s_1} \sigma_1 \\ \xi_2 + \frac{s_1 - s_2}{s_1 + s_2} x_j \xi_3 + \frac{y_j}{2s_2} \sigma_2 \end{pmatrix}) \geq 0$$
$$\begin{pmatrix} (x_i - x_j) & (y_i - y_j) \end{pmatrix} \begin{pmatrix} \frac{\sigma_1}{2s_1}(x_i - x_j) + \frac{s_1 - s_2}{s_1 + s_2}(y_i - y_j)\xi_3 \\ \frac{\sigma_2}{2s_2}(y_i - y_j) + \frac{s_1 - s_2}{s_1 + s_2}(x_i - x_j)\xi_3 \end{pmatrix} \geq 0$$
$$\begin{pmatrix} (x_i - x_j) & (y_i - y_j) \end{pmatrix} \begin{pmatrix} \frac{\sigma_1}{2s_1} & \frac{s_1 - s_2}{s_1 + s_2}\xi_3 \\ \frac{s_1 - s_2}{s_1 + s_2}\xi_3 & \frac{\sigma_2}{2s_2} \end{pmatrix} \begin{pmatrix} x_i - x_j \\ y_i - y_j \end{pmatrix} \geq 0$$
$$(p_i - p_j)^T \begin{pmatrix} \frac{\sigma_1}{2s_1} & \frac{s_1 - s_2}{s_1 + s_2}\xi_3 \\ \frac{s_1 - s_2}{s_1 + s_2}\xi_3 & \frac{\sigma_2}{2s_2} \end{pmatrix} (p_i - p_j) \geq 0 \tag{4.13}$$

The above equation is of quadratic form $w^T J w$ and J is symmetric matrix. Now for the above system to be positive semi-definite, J has to be positive semi-definite, which gives us a condition on the eigenvalues of J as follows:

$$\det(J - \lambda I) = 0$$
$$\det \begin{pmatrix} \frac{\sigma_1}{2s_1} - \lambda & \frac{s_1 - s_2}{s_1 + s_2}\xi_3 \\ \frac{s_1 - s_2}{s_1 + s_2}\xi_3 & \frac{\sigma_2}{2s_2} - \lambda \end{pmatrix} = 0$$
$$(\frac{\sigma_1}{2s_1} - \lambda)(\frac{\sigma_2}{2s_2} - \lambda) - (\frac{s_1 - s_2}{s_1 + s_2}\xi_3)^2 = 0$$
$$\lambda^2 - \lambda(\frac{\sigma_1}{2s_1} + \frac{\sigma_2}{2s_2}) + \frac{\sigma_1}{2s_1}\frac{\sigma_2}{2s_2} - (\frac{s_1 - s_2}{s_1 + s_2}\xi_3)^2 = 0$$

Thus,

$$\lambda = (\frac{\sigma_1}{s_1} + \frac{\sigma_2}{s_2}) \pm \sqrt{(\frac{\sigma_1}{s_1} + \frac{\sigma_2}{s_2})^2 - 4((\frac{\sigma_1}{s_1}\frac{\sigma_2}{s_2}) - (\frac{s_1 - s_2}{s_1 + s_2}\xi_3)^2)}$$

$$\lambda = (\frac{\sigma_1}{s_1} + \frac{\sigma_2}{s_2}) \pm \sqrt{(\frac{\sigma_1}{s_1} - \frac{\sigma_2}{s_2})^2 + 4(\frac{s_1 - s_2}{s_1 + s_2}\xi_3)^2} \tag{4.14}$$

The above eigenvalues must satisfy $\lambda \geq 0$ for the matrix to be positive semi-definite. Thus the following condition is obtained:

$$\lambda \geq 0$$

$$(\frac{\sigma_1}{s_1} + \frac{\sigma_2}{s_2}) \pm \sqrt{(\frac{\sigma_1}{s_1} - \frac{\sigma_2}{s_2})^2 + 4(\frac{s_1 - s_2}{s_1 + s_2}\xi_3)^2} \geq 0$$

$$(\frac{\sigma_1}{s_1} + \frac{\sigma_2}{s_2}) \geq \sqrt{(\frac{\sigma_1}{s_1} - \frac{\sigma_2}{s_2})^2 + 4(\frac{s_1 - s_2}{s_1 + s_2}\xi_3)^2} \tag{4.15}$$

Thus, a condition wherein if the size of the formation is increasing, the solution from the minimum-energy control law satisfies the inequality constraint is obtained. It was proved in Proposition 4: 4.1 that the minimum-energy control law satisfies the monotonic convergence inequality.

In the above proof, only cases when $\tilde{x}_4 > 0, \tilde{x}_5 > 0$ were dealt with. If $\tilde{x}_4 \leq 0, \tilde{x}_5 \leq 0$, i.e. when the shape in the abstract space is shrinking, there is limited guarantee that the minimum-energy solution satisfies the safety constraint. In other words, when this happens there may not be a non-zero control law v_i, that satisfies the inequalities without restrictions on change in orientations as shown by Proposition 4: 4.1. It is only in this condition that the system will reach an equilibrium away from the desired abstract state. This part has been demonstrated by us in the simulation part. Refer xxxxxx for visualization.

Also, there exists an abstract state x such that the minimum-energy solution given by the control law (3.11) satisfies any abstract state $\tilde{x}$, because Proposition 6: 4.2.1 mentioned above comes into application only when the robots are entering the colliding state as described by the equation $2\rho + \varepsilon_s$ (3.12).

4.3 Motion Planning

Motion planning is an important part that needs to be addressed while having large number of robots. Similar to the design of control law for an abstract state and then applying the transformation to obtain the equivalent control law in the inertial frame, the design of motion planning follows a similar approach. The abstract representation of the team of robots permits the planning of motion of robots to consider only the abstract state space rather than a traditional approach that scales with the number of robots. As the group of robots move from one position to the desired state, the formation or the ellipsoid encircling the robots change its shape along the course. Thus, the shape formation is deformable in nature which means the abstract representation is deformable in nature.

The approach taken in this section is to model the abstract representation as a deformable body and derive an energy metric associated with rotation, translation and deformation of the ensemble shape. The author has also included the ensemble contractions as this is a difficult case due to the increase in inter-robot communication.

Since, the abstract space is obtained by a surjective mapping ϕ of the configuration space, Riemannian metric on the abstract state x can be derived by an inner product on the associated Lie algebra, g. It is necessary to understand why a Riemannian metric is defined in this case. A Riemannian metric is a family of smoothly varying inner products on the tangent spaces of a smooth manifold. In our case of robot ensemble, we are considering ϕ to be a smooth mapping from the configuration space to an abstract space and each of these spaces to be smooth manifolds. Thus, Riemannian metric is an inner product on the tangent spaces, TQ and TM, where TQ is mapped to TM by $d\phi$. Thus, the abstract manifold M can be equipped with different Riemannian metrics.

A Riemannian metric on M is defined as a smooth family of inner products on the tangent space $TM \subset M$. The inner product of tangent vectors $\dot{g_1}, \dot{g_2} \in g \in SE(2)$ is obtained by left translation property:

$$\langle \dot{g_1} \cdot \dot{g_2} \rangle = \langle g^{-1}\dot{g_1}, g^{-1}\dot{g_2} \rangle_e \tag{4.16}$$

where $g^{-1}\dot{g_1}_i$ are tangent vectors at the identity element e. The above metric is an left invariant Riemannian metric. Following Zefran et. al. (1998) [6], the inertia tensor of a rigid body ans it's kinetic energy is used to define W_g:

$$W_g = \begin{pmatrix} mI_2 & 0 \\ 0 & I_{11} + I_{22} \end{pmatrix} \tag{4.17}$$

in the local frame B. But, the above condition is for rigid shapes. Since, the product structure $M = GxS$ was previously shown to be consisting of independent sets, the space model can be treated independently. The a constant metric $W_s = \alpha I_2$ is used to model the cost of the change in shape of the ensemble. Thus, the rate of change of the abstract shape in the local frame B is given by ζ and has the norm:

$$\|\zeta\| = \frac{1}{2}\zeta^T \begin{pmatrix} W_g & 0 \\ 0 & W_s \end{pmatrix} \zeta \tag{4.18}$$

which is well-defined everywhere on M.

Now, the potential energy associated with the deformation of shape must be made. The author has considered a simple approach to create an abstract model for the potential energy. Since, the deformation involves contraction and expansion of the shape, this process can be thought of as a reversible, adiabatic process where no energy is lost i.e when the formation compresses, the internal energy of the system increases and this energy is lost during the expansion of the formation making the net energy gained or lost equal to null. Hence, the process is said to be reversible. For such a reversible adiabatic process, pressure p and volume v are related to each other by the ratio of specific heats γ by the equation:

$$pv^\gamma = constant \tag{4.19}$$

and the work done to bring about a change in the volume from v_1 to v_2 leading to an increase in internal energy V is given by:

$$\Delta V = k\Big(\frac{1}{v_2^{\gamma-1}} - \frac{1}{v_1^{\gamma-1}}\Big) \tag{4.20}$$

where k is a constant.

In the case of planar robots, instead of volume v, the area of the concentration ellipsoid is used with unit depth. The area of the concentration ellipsoid is given by the standard formula for area of an ellipse $\pi\sqrt{s_1 s_2}$. If $s^0(N)$ is considered as a circular shape for N robots with zero potential energy, the radius of the circular shape $r_0(N)$ must increase with N. For simplicity, we take $r_0(N) = N^{\frac{\varepsilon}{2}}$. Thus, the potential energy is given by:

$$V(s) = \beta\Big(\frac{1}{(s_1 s_2)^{(\gamma-1)/2}} - \frac{1}{(r_0^2(N))^{\gamma-1}}\Big) \tag{4.21}$$

where β is a constant. Thus, the total energy associated with the configuration is given by :

$$E(g,s,\zeta) = \frac{1}{2}\zeta^T \begin{pmatrix} W_g & 0 \\ 0 & W_s \end{pmatrix} \zeta + V(s) \tag{4.22}$$

Using the above equation, cost of changes in configuration can be determined using a kinetic and potential energy.

The author has designed motion plan using discretization of the abstract space by adhering to the constraints defined by the obstacles. Motion plan determined by a sequence of desired abstract states is computer via Bellman-Ford search in a discretized abstract space.

Chapter 5

SIMULATION AND RESULTS

This section details about the simulation of the results of the research paper [2] in MATLAB. The simulation has been conducted for both point robots and non-holonomic robot defined by certain radius and axes length. The simulation considers the control of the formation to a desired abstract state for varying team sizes. The experimental results on real hardware have not been performed and instead the similar results have been simulated to check its validity.

5.1 Implementation Details

5.1.1 Point Robot

A point robot is controlled to a desired abstract state. The dynamics of the point robot is dependent on the control velocity u_i. The abstract state of the system at each instant of time is derived using the equations derived in the section 2 of the report. The robots are controlled to a desired time-invariant abstract state x^{des} of the system.

The controller design of each robot is based on the current position q_i of each robot, abstract state x of the robot ensemble. The current position and velocity of the adjacent robots are also considered to take into effect the collision avoidance in the system. The control law is decoupled, which means change in pose (position and orientation) does not affect the shape and vice-versa. The final desired abstract state is controlled using a proportional controller as given with below equations:

$$\begin{aligned}
\dot{\mu} &= K_\mu(\mu^d - \mu) \\
\dot{\theta} &= k_\theta(\theta^d - \theta) \\
\dot{s_1} &= k_{s_1}(s_1^d - s_1) \\
\dot{s_2} &= k_{s_2}(s_2^d - s_2)
\end{aligned}$$

where $K_\mu \in \mathbb{R}^{2x2}$ is a positive definite matrix and $k_\theta, k_{s_{1,2}} > 0$.

5.1.2 Non-Holonomic Robot

A differential drive robot is a non-holonomic robot as it has a constraint on its velocity that prevents it from moving in lateral direction instantaneously. The top-view of the robot which

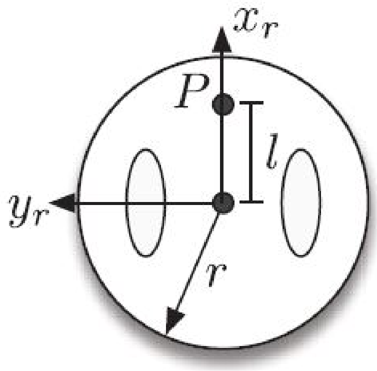


Figure 5.1: Differential drive robot being simulated.

is simulated is shown in Figure 5.1. Here, P is the reference point whose position is regulated by the vector fields and updated for every instant of time. r is the radius of the robot. l is the axes length. Model of a non-holonomic differential drive robot is given by:

$$\begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} \cos\theta_r & \sin\theta_r \\ -\frac{1}{\sin\theta_r} & \frac{1}{\cos\theta_r} \end{pmatrix} \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} \tag{5.1}$$

Here, (x, y) are the co-ordinates of the reference point P of the robot. This is updated for every time instant by the controller as q_i. v and ω are the linear and angular velocity of the robot respectively and θ_r is the orientation of the robot. Since, the radius of robot is r, a circle with this radius circumscribes the robot, a circle of radius $l+r$ centered at P, will contain all points of the robot. This point p and a line from P towards the direction of orientation is shown in the simulation.

5.2 Software Implementation

The simulation is performed in MATLAB. The code is written in the MATLAB. When compared to the results simulated by the author, we have taken a generic approach of a random distribution of the robots instead of uniform positioning of the robots as seen in Fig.5 of the paper [2]. However, one important detail to be considered is the robot formation should be such that $s_1 \neq 0$, $s_2 \neq 0$ and $s_1 \neq s_2$. In case of the two parameters being equal, the control law does not hold good as the states are not defined as mentioned in section 2.6.

The control law implemented in this paper considers the collision avoidance mechanism. In the software, we implement this by acquiring the position and velocity with respect to World Frame W for each robot and using this to solve the convex quadratic problem to get the optimum velocity solution $\hat{v}_i$ in the local frame B. The software package *lsqlin* in MATLAB is used to solve the quadratic optimization problem. The control velocity is determined at each instant of time and it is used to update the position of the robot every time interval based on Euler method of simulation.

The velocity is saturated to a maximum value to ensure that each robot is not subjected to a high velocity. This is based on the hardware considerations for the robot and for better convergence. The velocity determined by the minimum energy control law is considered to have a maximum of $0.05ms^{-1}$. Though this has not been explicitly mentioned by the author it can be confirmed from the graphs of the velocity plot in the report. The velocity of the robots determined by the convex problem is constrained to have a max of $0.1ms^{-1}$ as indicated by the author in [2].

The robot is modelled to have different radius r, and different safety region ε_s in the simulation. The collision avoidance of the robot swarm is taken into effect such that the robots are separated by the safe separation distance in each instances of time.

5.3 Results

Simulation is done for various configuration with different number of robots and desired abstract states of the formation as shown in the paper [2]. We simulate these results as implemented by the author in the following section. All the distances are in $meters(m)$, angles are in (rad).

1. Case 1: Refer Figure 5.2. N = 5, Centroid, μ = [3 3], θ = 0.9, s_1 = 0.25, s_2 = 0.15.

   The results below show that the robots converge to the desired state while avoiding collision. This can be observed from the plot of the magnitude of difference in position between each robot. Also, the plot of velocity depicts times at which the actual velocity is different from that found using the control law. The final configuration indicates the robot position with the abstract state variables of the robot being equal to the desired state. This can be observed from the centroid ellipse of the actual robot configuration and the desired state.

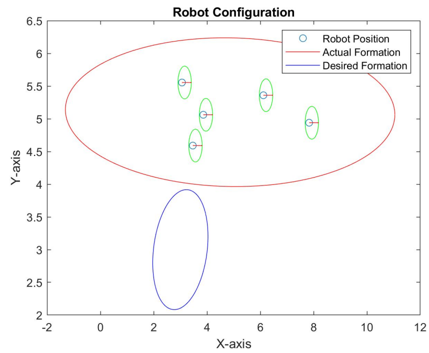
2. Case 2: Refer Figure 5.3. N = 10, Centroid, μ = [3 3], θ= 0.9, s_1 = 0.6, s_2 = 0.3. The results follow a similar behavior as case 1. The robots converge to the desired state while satisfying the collision avoidance constraint. However, note that the inter robot communications increase a lot as they approach the desired state as expected.

3. Case 3: Refer Figure 5.4. N = 20, Centroid = [3 3], θ = 0.9, s_1 = 0.3, s_2 = 0.15. In case of larger number of robots, it is observed that the system converges to the desired final state while not satisfying the collision avoidance criterion. We tried for the parameters as specified by the author. This condition failed. Thus, we tried to increase the safety distance, the results were slightly better, however, the collision free movement could not be obtained. Thus, we could infer that the dynamics of the non-holonomic robot implemented by us must be more accurate to avoid collision.

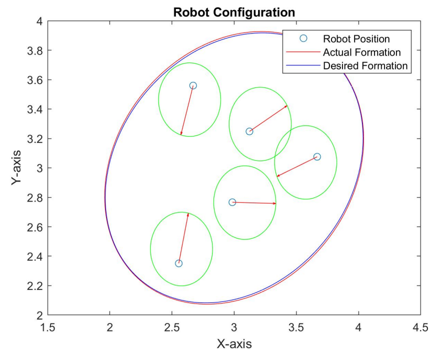
4. Case 4: Refer Figure 5.5. N= 10, Centroid = [3 3], θ = 0.9, s_1 = 0.3, s_2 = 0.15. In this case, it is observed that the system does not converge to the desired state and there is an error in the final position's formation. While there is an error in the system, it is still observed that the collision avoidance is satisfied in the behavior. The final configuration plot shows that the desired ellipse is not collinear with the centroid of the final position of swarm which shows the error from the final desired state.
   The inference from the above case is that the final ensemble cannot achieve a desired position that requires contraction of the system due to the physical system description. This is a limitation to this methodology. However, the system behavior is safe as collision is avoided.

5. Case 5: Refer Figure 5.6. $N = 7$, Centroid [2 1], $\theta = 0.5$, $s_1 = 0.5$, $s_2 = 0.25$ In this case the system converges to the desired state avoiding collisions and converging monotonically.

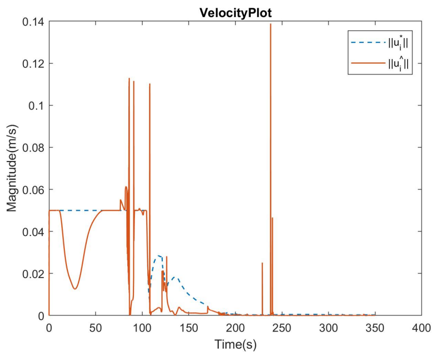
6. Case 6: Refer Figure 5.7. N = 7, Centroid [0.5 0], $\theta = 0$, $s_1 = 0.4$, $s_2 = 0.2$ The system converges to the desired state as expected but with a slight error in the final formation. This is due to the the constaint on the physical description of the system for which these sets of robots cannot achieve 0 error state.



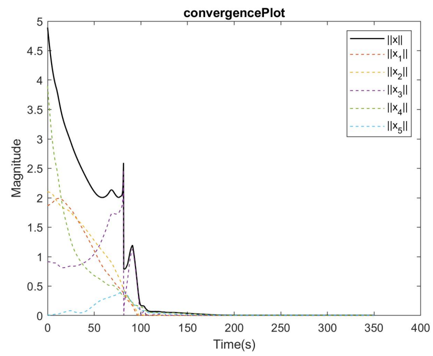
(a) Initial state of the ensemble



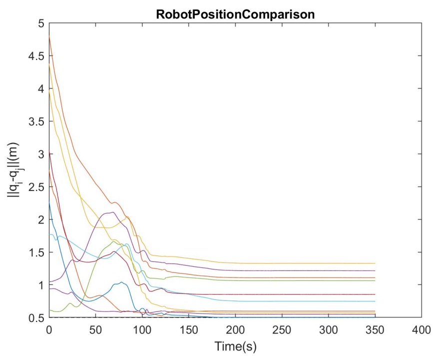
(b) Final state of the ensemble



(c) Velocity plot

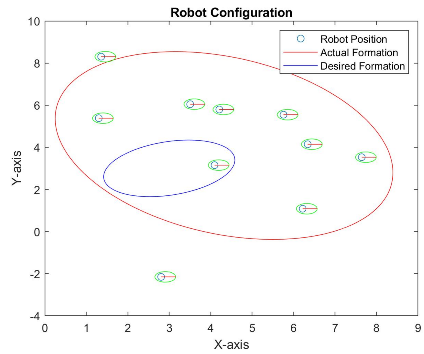


(d) Convergence Plot error $\tilde{x}$

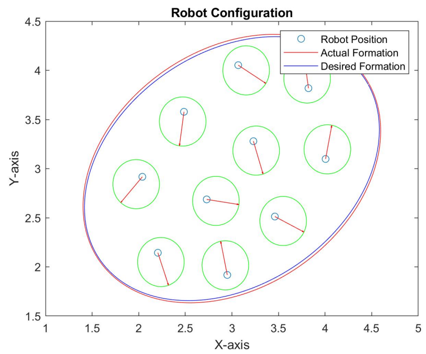


(e) Robot position Convergence Plot

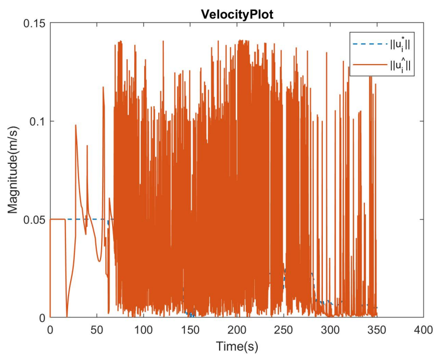
Figure 5.2: Results Case 1



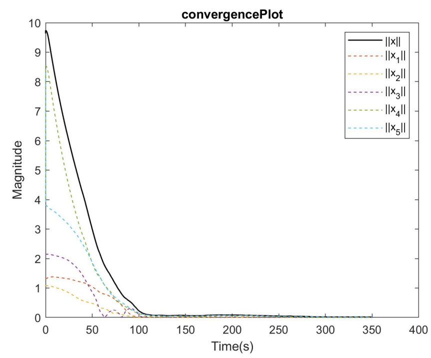
(a) Initial state of the ensemble



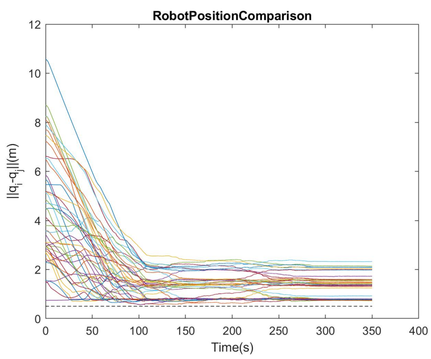
(b) Final state of the ensemble



(c) Velocity plot

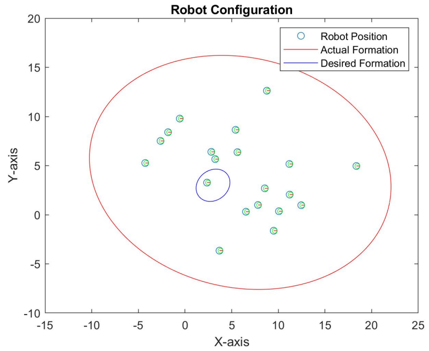


(d) Convergence Plot error $\tilde{x}$

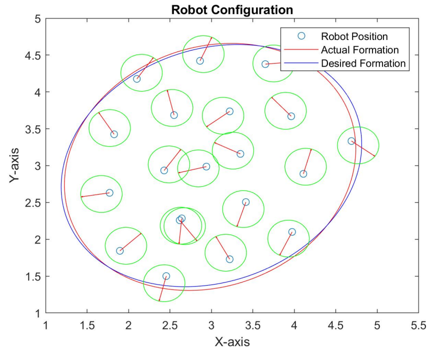


(e) Robot position Convergence Plot

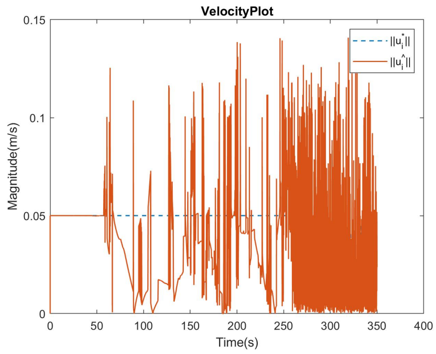
Figure 5.3: Results Case 2



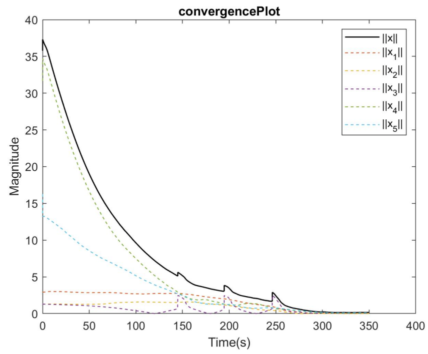
(a) Initial state of the ensemble



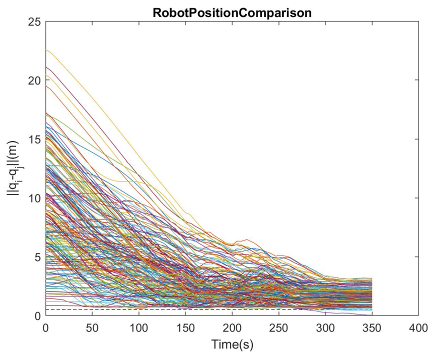
(b) Final state of the ensemble



(c) Velocity plot

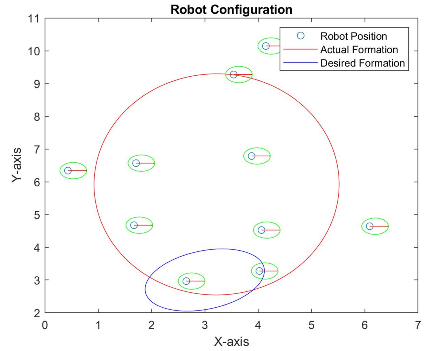


(d) Convergence Plot error $\tilde{x}$

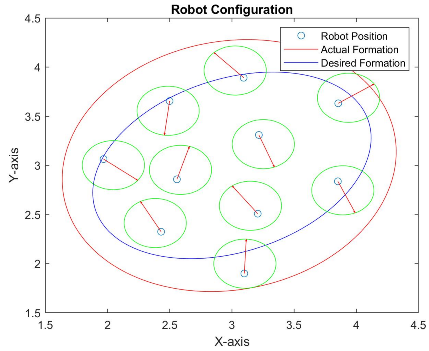


(e) Robot position Convergence Plot

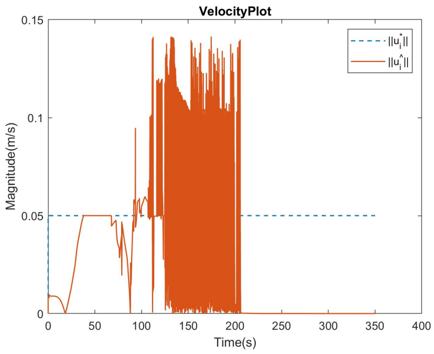
Figure 5.4: Results Case 3



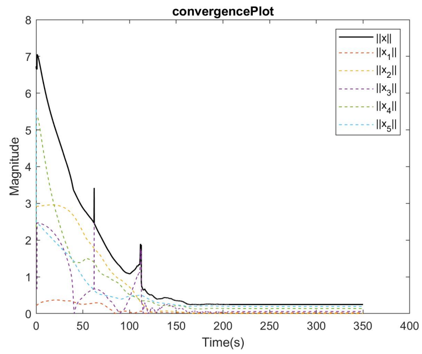
(a) Initial state of the ensemble



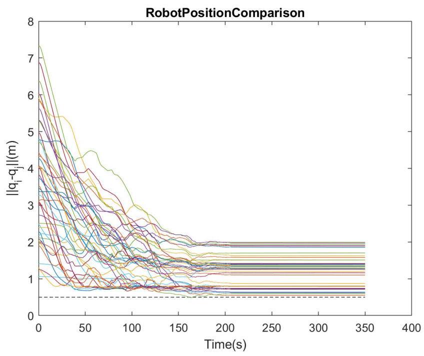
(b) Final state of the ensemble



(c) Velocity plot

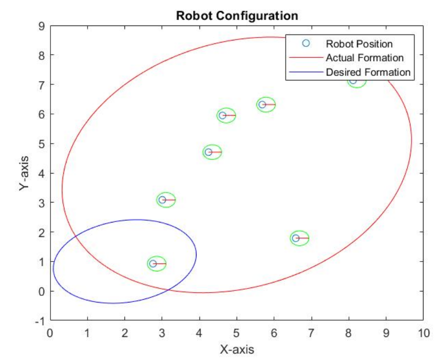


(d) Convergence Plot error $\tilde{x}$

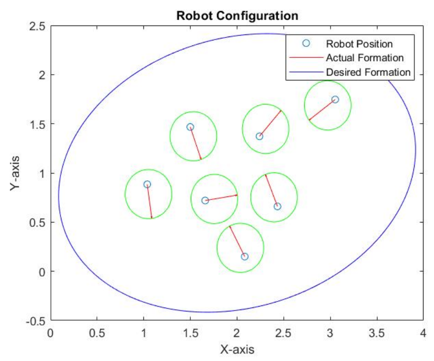


(e) Robot position Convergence Plot

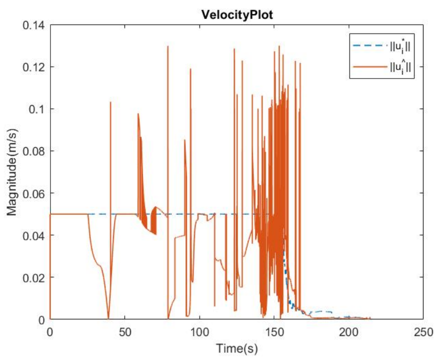
Figure 5.5: Results Case 4



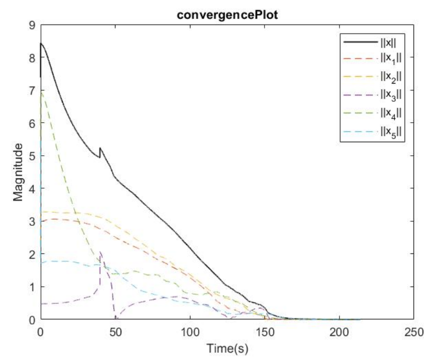
(a) Initial state of the ensemble



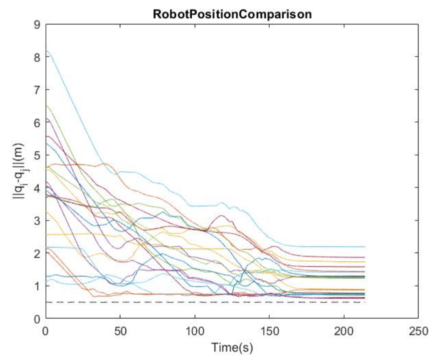
(b) Final state of the ensemble



(c) Velocity plot

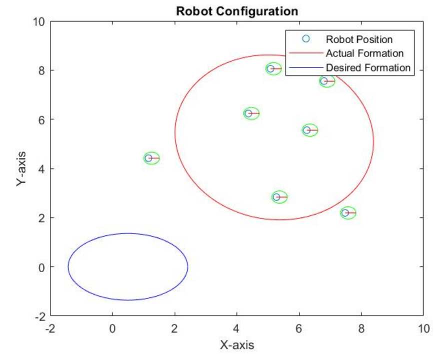


(d) Convergence Plot error $\tilde{x}$

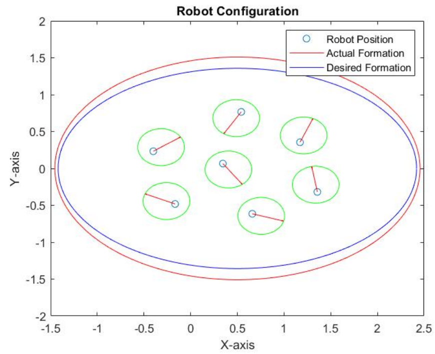


(e) Robot position Convergence Plot

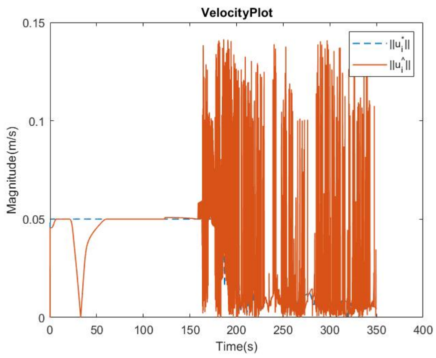
Figure 5.6: Results Case 5



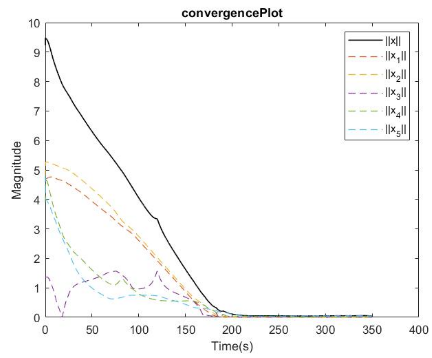
(a) Initial state of the ensemble



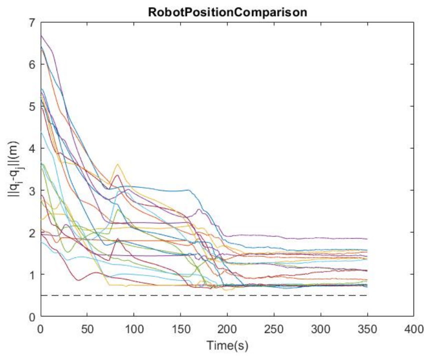
(b) Final state of the ensemble



(c) Velocity plot



(d) Convergence Plot error $\tilde{x}$



(e) Robot position Convergence Plot

Figure 5.7: Results Case 6

Chapter 6

CONCLUSION

The report presents an approach to defining the shape and formation of an ensemble of robots that is independent of the ordering and number of robots. The algorithm detailed in this report considers the physical structure of the robot, while ensuring collision avoidance between the members of the team. The results have been shown by simulation of the algorithm in MATLAB software environment for differential drive robots and this shows the effectiveness of the algorithm for non-holonomic robots.

Bibliography

[1] Belta, C. and Kumar, V. (2004). Abstraction and control for groups of robots. IEEE Transactions on Robotics, 20(5):865875.

[2] Michael, N. and Kumar, V. (2008). Controlling shapes of ensembles of robots of finite size with nonholonomic constraints. Robotics: Science and Systems, Zurich, Switzerland.

[3] Michael, N., Fink, J. and Kumar, V. (2007). Controlling a team of ground robots via an aerial robot. Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, San Diego, CA, pp. 965970.

[4] Michael, N., Belta, C. and Kumar, V. (2006). Controlling three dimensional swarms of robots. Proceedings of the IEEE International Conference on Robotics and Automation, Orlando, FL, pp. 964969.

[5] https://people.richland.edu/james/lecture/m170/tbl-chi.html.

[6] Zefran,M., Kumar, V. and Croke, C. B. (1998). On the generation of smooth three-dimensional rigid body motions. IEEE Transactions on Robotics and Automation, 14(4): 576589.

APPENDICES

Appendix A

MATLAB Code for Simulation of Swarm Robots with Non-Holonomic Constraints

```
function robotSwarmFormation()
% Constants
global N dT  KU_COEFF KS1_COEFF KS2_COEFF KT_COEFF

N = 5;          % number of bots
dT = .01;       % timestep length (position is updated each step)

% Control gains - This is taken from the 2004 paper. These are the gains
% for the individual abstract state variable used in the simulation of the
% paper
KU_COEFF = [2 0; 0 2];
KS2_COEFF = 2;
KT_COEFF = 2;

% Counter to accumulate the values of the data after each time interval
PLOT_COUNTER = 1;


%%%%%%%%%%%%%
% Main loop %
%%%%%%%%%%%%%
outpath = pwd;
% outputVideo = VideoWriter(fullfile(outpath,'SimulationVideo.mp4'),'MPEG-4');
% open(outputVideo);

% Place the robot swarm in the space
bots = distributeBots(N);

```

```
% Time and posPlot to accumulate values
Time(1) = 0;
KS1_COEFF = 2;
posPlot(N,1) = struct;

for i=1:N
    posPlot(i).qx(1) = bots(i).q(1) ;
    posPlot(i).qy(1) = bots(i).q(2) ;
    posPlot(i).uStarX(1) = 0 ;
    posPlot(i).uStarY(1) = 0 ;
    posPlot(i).uCapX(1) = bots(i).u(1) ;
    posPlot(i).uCapY(1) = bots(i).u(2) ;
end

PLOT_COUNTER = PLOT_COUNTER + 1;

%Initialize the abstract space
uCentroid = [0 0].';
theta = 0;
s1 = 0;
s2 = 0;

%Initialize the matrices to calculate shape variables
E1 = [0 1; 1 0];
E2 = [1 0; 0 -1];
E3 = [0 -1; 1 0];


%Initializing the desired position variables
uCentroidD = [3 ; 3];
s1D = 0.25;
s2D = 0.15;
thetaD = 0.9 ;

%Distance between bots - Taken from simulation of the current research
%paper
botRadius = 0.15;
botAxleLength = 0.1;
safeDist = 0.1 ;
sepDist = 2*(botRadius + botAxleLength) + safeDist;

%Initialize the configuration
[uCentroid,theta,s1,s2] = abstractSpace(bots);

figure;
```

```
drawElipseBoundary(bots,uCentroid,theta,s1,s2,uCentroidD,thetaD,s1D,s2D,botRadius, botAxleLength);
%Create a figure handle which we like to capture as a movie
figure;
% F = getframe(gcf);

%Initial position of the robots with the elipse
drawElipseBoundary(bots,uCentroid,theta,s1,s2,uCentroidD,thetaD,s1D,s2D,botRadius, botAxleLength);
% writeVideo(outputVideo,getframe(gcf));

%Initialize plot variables
uPlotx(1) = uCentroidD(1) - uCentroid(1) ;
uPloty(1) = uCentroidD(1) - uCentroid(2) ;
thetaPlot(1) = thetaD - theta ;
s1Plot(1) = s1D - s1;
s2Plot(1) = s2D - s2;

% This variable represents the complete state of the system
xPlot(1) = norm([uPlotx(1) ; uPloty(1) ; thetaPlot(1) ; s1Plot(1) ; s2Plot(1)],2);

% step counter for every intervals of time
stepCounter = 0;
keepLooping = true;

%Values to contain the final velocity calculated from the convex
%optimization problem
uxMax = [0.1 ; 0.1];
kVel = [0 ; 0];
nVel = 0;

%Values to contain the min energy control law velocity. This is in
%accordance with the paper where the min energy control law velocity is
%having magnitude maximum of 0.05.
uxMaxCtrlLaw = 0.05;

conditionFailure = 0 ;

while (true == keepLooping && (350*1/dT >= stepCounter))

    % Calculating the abstract state variables
    [uCentroid,theta,s1,s2] = abstractSpace(bots);
```

```
    R = [cos(theta) -sin(theta); sin(theta) cos(theta)];
    H1 = eye(2) + R^2*E2;
    H2 = eye(2) - R^2*E2;
    H3 = R^2*E1 ;
    g = [[R uCentroid];0 0 1];

    %Check if the desired formation has been reached
    if(isequal(round((uCentroidD(1) - uCentroid(1)),3),0) && isequal(round((uCentroidD(2) - uCentroid(2)),3),0) && isequal(round((s1D-s1),3),0) && isequal(round((s2D-s2),3),0) && isequal(round((thetaD-theta),4),0))
        break;
    end

    %Calculate the error gains for each of the output
    dCentroid = KU_COEFF*(uCentroidD - uCentroid);
    dTheta = KT_COEFF*(thetaD - theta);
    dS1 = KS1_COEFF*(s1D - s1);
    dS2 = KS2_COEFF*(s2D - s2);

    for robotInd1=1:N
        position = bots(robotInd1).q.'; % Current position of the robot
        %Calculation of velocity using min energy control law
        velocity = dCentroid + ((s1-s2)*H3*(position - uCentroid)*dTheta / (s1+s2))....
            + (H1*(position - uCentroid)*dS1 / 4*s1) + (H2*(position - uCentroid)*dS2/4*s2);
        % ui*

        %Scaling the values of the min energy ctrl velocity to 0.05ms-1
        nVelCtrlLaw = max(1,norm(velocity,2)/uxMaxCtrlLaw);

        velocity = velocity / nVelCtrlLaw ;

        % Converting u to v using R and also position w.r.t moving frame
        movPosition = R.'*(position - uCentroid); %pi
        movVelocity = R.'*velocity; %vi*

        %Inequality constraint for asymptotic convergence
        % stateTilde is the error of the state
        stateTilde = [uCentroidD - uCentroid;thetaD-theta;s1D-s1;s2D-s2];
```

```
        % Gamma is the transformation matrix from moving frame
            to abstract
        % space
        Gamma = [g zeros(3,2);zeros(2,3) eye(2)];

        % 5 by 1 matrix used in the monotonic convergence
           criterian
        val = [eye(2);(1/s1-s2)*movPosition.'*E1;movPosition
           .'*(eye(2)+E2);movPosition.'*(eye(2)-E2)];

        %Gain Matrix - 5*5 matrix
        GainMat = [KU_COEFF(1,:) 0 0 0; KU_COEFF(2,:) 0 0 0; 0
            0 KT_COEFF 0 0; 0 0 0 KS1_COEFF 0; 0 0 0 0
           KS2_COEFF];

        Acondition1 = stateTilde.'*GainMat*Gamma*val;

        %Inequality constraint to saturate the maximum
           velocity of the robot
        % calculated using convex optimization to max of 0.1ms
           -1
        % u = Rv
        Acondition2 = [1 0] * R ;
        Acondition3 = [0 1] * R;

        AMatCondition =[-Acondition1 ; Acondition2 ;
           Acondition3];
        BMatCondition = [0; 0.1; 0.1];

        %Check for conditions when the robots are within
           collision distance.
        %This is when the ccollision avoidance constrained is
           applied for the
        %robots
        for robotInd2=1:N
            % calculate the position of each robot and compare
                against
            % current
            if(robotInd2 ~= robotInd1)
                bot1Position = R.'*(bots(robotInd1).qN.' -
                   uCentroid); %p1
                bot2Position = R.'*(bots(robotInd2).qN.' -
                   uCentroid); %p2 - Old

```

```
        bot2Velocity = R.'*(bots(robotInd2).uN.');%v2 - New
        %the calculated delta value
        delta = norm(bot2Position - bot1Position, 2);

        if(delta <= sepDist)
            % Condition for the collision avoidance
            Acondition4 = (bot1Position - bot2Position).';
            Bcondition4 = ((bot1Position - bot2Position).'*bot2Velocity);
            AMatCondition = [AMatCondition ; -Acondition4];
            BMatCondition =  [BMatCondition ; -Bcondition4];
        end
    end
end

% Solving for optimal velocity based on previous condition
opts1 = optimset('display','off');
velocityCap = lsqlin(sqrt(2)*eye(2),sqrt(2)*movVelocity,AMatCondition,BMatCondition,[],[],[],[],[],opts1); %vicap

% There are instances in which the lsqlin function fails. This is a
% check for the failure to debug the system
if(round(AMatCondition*velocityCap,3) > round(BMatCondition,3))
    AMatCondition*velocityCap - BMatCondition
    conditionFailure = conditionFailure + 1;
end

% velocity with respect to the world frame
velWorldFrame = R*velocityCap ; %uicap

%As per the paper we will contain the velocity at max cap velocity
%for the robot
kVel(1) = max(1, abs(velWorldFrame(1))/uxMax(1));
kVel(2) = max(1, abs(velWorldFrame(2))/uxMax(2));

nVel = max(kVel(1),kVel(2));

```

```
        velWorldFrame = velWorldFrame / nVel ;

        % Update the position of the robot based on Euler method for simulation
        % Modelling the position based on the differential drive robot
        % model
        updateDifferentialDrivePos(velWorldFrame, botAxleLength, robotInd1);

        bots(robotInd1).uStar = velocity.' ; %ui*
    end

    %Update each of the robot position
    for robotInd1=1:N

        %Update the position and velocity variables of the robots
        bots(robotInd1).q = bots(robotInd1).qN ;
        bots(robotInd1).u = bots(robotInd1).uN;
        bots(robotInd1).tr = bots(robotInd1).trN ;

        % Accumulate the plot variables
        % Position of robots
        posPlot(robotInd1).qx(PLOT_COUNTER) = bots(robotInd1).q(1);
        posPlot(robotInd1).qy(PLOT_COUNTER) = bots(robotInd1).q(2);
        % Minimum energy control velocity
        posPlot(robotInd1).uStarX(PLOT_COUNTER) = bots(robotInd1).uStar(1);
        posPlot(robotInd1).uStarY(PLOT_COUNTER) = bots(robotInd1).uStar(2);
        % Control velocity input based on convex optimization
        posPlot(robotInd1).uCapX(PLOT_COUNTER) = bots(robotInd1).u(1);
        posPlot(robotInd1).uCapY(PLOT_COUNTER) = bots(robotInd1).u(2);

    end

    stepCounter = stepCounter+1;

    %Accumulate the data values for the plot
    Time(PLOT_COUNTER) = (Time(1) + dT*stepCounter);
    uPlotx(PLOT_COUNTER) = (uCentroidD(1) - uCentroid(1)) ;
```

```
    uPloty(PLOT_COUNTER) = (uCentroidD(2) - uCentroid(2)) ;
    thetaPlot(PLOT_COUNTER) = (thetaD - theta) ;
    s1Plot(PLOT_COUNTER) = (s1D-s1);
    s2Plot(PLOT_COUNTER) = (s2D - s2);
    xPlot(PLOT_COUNTER) = norm([uPlotx(PLOT_COUNTER) ; uPloty(PLOT_COUNTER) ; thetaPlot(PLOT_COUNTER) ; s1Plot(PLOT_COUNTER) ; s2Plot(PLOT_COUNTER)],2);

    PLOT_COUNTER = PLOT_COUNTER+1;

    if(0 == isnan(s1) && 0 == isnan(s2) )
        s1 , s2, uCentroid, theta
    end

    if(mod(stepCounter,20) == 0)
        drawElipseBoundary(bots,uCentroid,theta,s1,s2,uCentroidD,thetaD,s1D,s2D,botRadius, botAxleLength);
%           writeVideo(outputVideo,getframe(gcf));
    end

end

%Generate the data for difference in bot position
l=1;
for j=1:N
    for k=(j+1):N
        for plotCount=1:PLOT_COUNTER-1
            tempMatVal = [posPlot(j).qx(plotCount) - posPlot(k).qx(plotCount) ; posPlot(j).qy(plotCount) - posPlot(k).qy(plotCount)];
            posDiffPlot(l).q(plotCount) = norm(tempMatVal,2);

        end
        l = l+1;
    end
end

%Generate the data for the velocity plot
for j=1:N
    for plotCount=1:PLOT_COUNTER-1
    velStarPlot(j).u(plotCount) = norm([posPlot(j).uStarX(plotCount) posPlot(j).uStarY(plotCount)],2);
    velCapPlot(j).u(plotCount) = norm([posPlot(j).uCapX(plotCount) posPlot(j).uCapY(plotCount)],2);
    end
end
```

```
% Draw the final elipse position
drawElipseBoundary(bots,uCentroid,theta,s1,s2,uCentroidD,thetaD,s1D,s2D,botRadius, botAxleLength);
% writeVideo(outputVideo,getframe(gcf));

% close(ouVtputVideo);

%We start Plotting our parameters here
% Plot of the error in the state variables
figure ;

plot1 = plot(Time,xPlot,Time, abs(uPlotx),'--',Time, abs(uPloty),'--',Time, abs(thetaPlot),'--', Time, abs(s1Plot), '--',Time, abs(s2Plot), '--','linewidth', 0.7);
plot1(1).LineWidth = 1;
plot1(1).Color = 'black';
xlabel('Time(s)');
ylabel('Magnitude');
legend('||x||','||x_1||','||x_2||','||x_3||','||x_4||','||x_5||');
title('convergencePlot');
hold on;


figure ;
plot(Time,2*(botRadius + botAxleLength)*ones(size(Time)),'--','Color','black');
hold on;
for i=1:l-1
    plot(Time,posDiffPlot(i).q);
    hold on;
end
xlabel('Time(s)');
ylabel('||q_i-q_j||(m)');
title('RobotPositionComparison');

hold on;

figure ;
plot(Time, velStarPlot(1).u,'--', Time, velCapPlot(1).u,'linewidth',1);
xlabel('Time(s)');
ylabel('Magnitude(m/s)');
title('VelocityPlot');
legend('||u_i^{*}||','||u_i^{\^}||');
```

```
hold on;

figure;
drawElipseBoundary(bots,uCentroid,theta,s1,s2,uCentroidD,thetaD,s1D,s2D,botRadius,botAxleLength);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Bot initialization                                                              %
% Description - This function distributes the robots in Euclidean space as%
% a normal distribution. The separation between each robot should be      %
% greater than the initial configuration                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function bots = distributeBots(N)

        %Structure variables:
        %q - holds the position w.r.t the world frame
        %u - Velocity w.r.t the world frame
        %qN - Holds the new position of the robots that has to be updated
        bots(N,1) = struct('q',[0; 0],'u',[0; 0].','qN',[0 ; 0],'uN',[0 ; 0],'uStar',[0 ; 0],'tr',0,'trN',0);


        % Choosing Random distribution of the robots where the robots are
        % separated at a distance greater thatn the safe separation distance
        % between each of them. We choose
        % an arbitrary value of the mean and the standard deviation

        randLoop = true;
        while(randLoop)
            counter = 0;
            % Standard deviation is equal to robot count / 4
            A = normrnd(5,max(1,N/4),[2,N]);

            for robotInd1=1:N
```

```
                for robotInd2=1:N
                    if(robotInd1 ~= robotInd2)
                        if(norm((A(:,robotInd1) - A(:,robotInd2)),2) < 0.6)
                            counter = counter+1;
                        end
                    end
                end
            end

            if(counter == 0)
                %Print the position of the robots
                A
                randLoop = false;
            end
        end

        for robotInd=1:N
            % place agent, Initializing all states of the robots
            bots(robotInd).q = A(1:2,robotInd).';
            bots(robotInd).qN = A(1:2,robotInd).';
            bots(robotInd).u = [0 0];
            bots(robotInd).uN = [0 0];
            bots(robotInd).uStar = [0 0];
            bots(robotInd).tr = 0 ;
            bots(robotInd).trN = 0 ;
        end
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Name - Abstract Space                                                              %
% Description - Computes the abstract state variables of the formation     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    function [aCentroid,aTheta,aS1,aS2] = abstractSpace(bots)
        E1 = [0 1; 1 0];
        E2 = [1 0; 0 -1];
        E3 = [0 -1; 1 0];

        tThetaY = 0;
```

```
        tThetaX = 0;
        tS1 = 0;
        tS2 = 0;
        tempCentroid = 0;

        %Calculate Centroid
        for robotInd =1:N
            position = bots(robotInd).q.';
            tempCentroid = tempCentroid + position ;
        end
        aCentroid = 1/N * tempCentroid ;

        %Calculate Orientation
        for robotInd = 1:N
            position = bots(robotInd).q.';
            tThetaY = tThetaY + (position - aCentroid).'*E1*(position - aCentroid);
            tThetaX = tThetaX + (position - aCentroid).'*E2*(position - aCentroid);
        end
            aTheta = atan2(tThetaY , tThetaX)/2 ;
            R = [cos(theta) -sin(theta); sin(theta) cos(theta)];
            H1 = eye(2) + R^2*E2;
            H2 = eye(2) - R^2*E2;
            H3 = R^2*E1 ;

        for robotInd=1:N
            position = bots(robotInd).q.';
            tS1 = tS1 + ((position - aCentroid).'*H1*(position - aCentroid)) ;
            tS2 = tS2 + ((position - aCentroid).'*H2*(position - aCentroid)) ;
        end

        aS1 = tS1 / (2*(N-1)) ;
        aS2 = tS2 / (2*(N-1)) ;

    end

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Name - drawElipseBoundary                                                                      %
```

```
% Description - Draw elipse boundary around the robot formation and the %
% final desired positon                                                  %
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function drawElipseBoundary(aBots, aCentroid , aTheta, aS1, aS2 , aCentroidD, aThetaD, aS1D, aS2D, botRadius, botAxleLength)
        % Plot the robot positions and the ellipsoid
        X = [];
        t=0:0.01:2*pi;

        for robotInd=1:N
            X = [X [aBots(robotInd).q(1);aBots(robotInd).q(2)]];
        end

        if aS1>aS2
            % Multiply [acos(t); bsin(t)] by R. Taking confidence parameter as 1
            x1 = aCentroid(1) + sqrt(9.2103*aS1)*cos(t)*cos(aTheta) - sqrt(2*aS2)*sin(t)*sin(aTheta);
            y1 = aCentroid(2) + sqrt(9.2103*aS2)*sin(t)*cos(aTheta) + sqrt(2*aS1)*cos(t)*sin(aTheta);
        else
            % Multiply [acos(t); bsin(t)] by R. Taking confidence parameter as 1
            x1 = aCentroid(1) + sqrt(9.2103*aS2)*cos(t)*cos(aTheta) - sqrt(2*aS1)*sin(t)*sin(aTheta);
            y1 = aCentroid(2) + sqrt(9.2103*aS1)*sin(t)*cos(aTheta) + sqrt(2*aS2)*cos(t)*sin(aTheta);
        end

        %Plot the desired elipse position

        if aS1D>aS2D
            % Multiply [acos(t); bsin(t)] by R. Taking confidence parameter as 1
            x2 = aCentroidD(1) + sqrt(9.2103*aS1D)*cos(t)*cos(aThetaD) - sqrt(2*aS2D)*sin(t)*sin(aThetaD);
            y2 = aCentroidD(2) + sqrt(9.2103*aS2D)*sin(t)*cos(aThetaD) + sqrt(2*aS1D)*cos(t)*sin(aThetaD);
        else
```

```
            % Multiply [acos(t); bsin(t)] by R. Taking
              confidence parameter as 1
            x2 = aCentroidD(1) + sqrt(9.2103*aS2D)*cos(t)*cos(
              aThetaD) - sqrt(2*aS1D)*sin(t)*sin(aThetaD);
            y2 = aCentroidD(2) + sqrt(9.2103*aS1D)*sin(t)*cos(
              aThetaD) + sqrt(2*aS2D)*cos(t)*sin(aThetaD);
        end

%          labels = {'1','2','3','4','5','6','7','8','9','10'};
        plot(X(1,:),X(2,:),'o');
%          text(X(1,:),X(2,:),labels,'VerticalAlignment','
    bottom','HorizontalAlignment','right');
        title('Robot Configuration')
        xlabel('X-axis')
        ylabel('Y-axis')
        hold on
        plot(x1,y1,'r',x2,y2,'b');
        for robotInd=1:N
            botCenter = aBots(robotInd).q;
            botOrientation = aBots(robotInd).tr;
            rotMat = [cos(botOrientation) -sin(botOrientation)
              ;sin(botOrientation) cos(botOrientation)];
%              rotMat = [cos(botOrientation) sin(botOrientation
    ) botRadius*cos(botOrientation); -sin(botOrientation) cos(
    botOrientation) botRadius*sin(botOrientation); 0 0 1];
            lineEnd = botCenter + [(botRadius+botAxleLength)*
              cos(botOrientation) (botRadius+botAxleLength)*
              sin(botOrientation)];
%              lineEnd = lineEnd(1:2,1).';
            % Defining circles around each robot
            x = aBots(robotInd).q(1)+ botAxleLength*cos(
              botOrientation) + (botRadius+botAxleLength)*cos(
              t);
            y = aBots(robotInd).q(2)+ botAxleLength*sin(
              botOrientation) + (botRadius+botAxleLength)*sin(
              t);
            plot(x,y,'g');
            quiver(botCenter(1,1),botCenter(1,2),lineEnd(1,1)+
              botAxleLength*cos(botOrientation) - botCenter
              (1,1), lineEnd(1,2)+botAxleLength*sin(
              botOrientation) - botCenter(1,2),0,'Color','red'
              );

        end
        legend('Robot Position','Actual Formation','Desired
          Formation');
```

```
        hold off
    end

    function updateDifferentialDrivePos(velWorldFrame, axleLength, robotInd)
        linVel = 0 ;
        angVel = 0 ;

        botOrientation = bots(robotInd).tr ; %New Orientation
        rotMat = [cos(botOrientation) sin(botOrientation) ; -sin(botOrientation)/axleLength cos(botOrientation)/axleLength];

        velMat = rotMat*velWorldFrame ;

        linVel = velMat(1);
        angVel = velMat(2);

        %Update the kinematic position of the robots. We use the same model
        %that is used in the robotic simulator toolbox to update the
        %position
        dx = dT * linVel * cos(botOrientation);
        dy = dT * linVel * sin(botOrientation);

        dtr = dT * angVel ;

        %Update the new positions and new velocities to which the robots
        %have to move and the new orientation of the robots
        bots(robotInd).qN(1)  = bots(robotInd).q(1) + dx ;
        bots(robotInd).qN(2) = bots(robotInd).q(2) + dy;

        bots(robotInd).uN = velWorldFrame.';

        bots(robotInd).trN = bots(robotInd).tr + dtr ;
    end

end
```