Summer 8-2013

Decentralized Collision Avoidance

Jayasri K. Janardanan
University of Nebraska-Lincoln, jayasri_jan@yahoo.com

Decentralized Collision Avoidance

by

Jayasri K Janardanan

A THESIS

Presented to the Faculty of

The Department of Computer Science at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Carrick Detweiler

Lincoln, Nebraska

August, 2013

Decentralized Collision Avoidance

Jayasri K Janardanan, M.S.

University of Nebraska, 2013

Adviser: Carrick Detweiler

Autonomous Robots must carry out their tasks as independently as possible and each robot may be assigned different tasks at different locations. As these tasks are being performed, the robots have to navigate correctly such that the assigned tasks are completed efficiently, while also avoiding each other and other obstacles. To accomplish effective navigation, we must ensure that the robots are calibrated to avoid colliding with any kind of object on its path. Each robot has to sense the obstacles on its path and take necessary corrective measure to avoid those obstacles. In a situation with multiple robots, robots may cross each other's paths and thus algorithms have to be developed to ensure collision avoidance among them.

Collision avoidance among multiple robots has been studied extensively over many years. In this thesis, we investigate the Reciprocal n-body Collision Avoidance Algorithm (RCAA) where collision avoidance among multiple robots is addressed. One advantage of RCAA over other techniques is the decentralized approach that allows robots to take collision- avoidance decisions by themselves using only velocity and position of the nearby robots that are along its trajectory. Though this method is widely used, a major limitation is the assumption of perfect sensing, which is not a guaranteed behavior in real environments. In real world scenarios, erroneous measurements may be obtained during which the RCAA is not capable of ensuring perfect collision avoidance.

This limitation in the RCAA needs to be addressed and in this thesis, we have devised a method to address this using particle filters.

A particle filter is appended to the RCAA to sample velocities and thereby provide the robots with more options to avoid coming in the path of each other. A simulation program has been developed to implement the entire system showcasing different scenarios where the introduction of particle filter has made the system more stable as it ensures a more streamlined and efficient collision avoidance among robots.

I dedicate this thesis to my parents

Acknowledgements

I would like to take this opportunity to sincerely thank my advisor Dr.Carrick Detweiler for his constant support and encouragement. I would like to express my deepest appreciation to him for introducing me into this area of research and for motivating me throughout. Also, I would like to express my gratitude to Dr.Jitender Deogun and Dr.Ashok Samal for agreeing to be on my committee. Furthermore, I would like to thank my loved ones for keeping me together.

Table of Contents

List of Figures

Chapter 1

Introduction

Research on mobile robots has gained sufficient traction over the years as the robots are being investigated for use in different kinds of missions. Significant work has been carried out in various disciplines like underwater research, studying the flock of birds, collecting data for ecological study, handling hazardous ammunitions and fighting fire hazards to name a few. The scope and the opportunities that have come up in this area in the past few years have been tremendous. In all these applications, an important requirement is that the robots should be able to navigate easily in any kind of environment to carry out the task assigned to them. In order to do the same, the robots should localize and move without colliding with any kind of obstacle on its way.

Collision is a fundamental problem in mobile robotics and therefore collision avoidance becomes an important research area. Over the years, there have been various approaches that have been formulated, analyzed and discussed. The obstacles that are considered in such research can be moving or static and are usually other robots that are involved in that operation. The research presented here focuses only on homogeneous robots where each one is considered as a moving obstacle with respect to the rest. In real world applications, robots work in uncertain environments and therefore in motion planning, one of the main objectives of the robot would be to avoid any kind of unexpected object on its way while progressing towards its goal. In a decentralized approach, the responsibility of collision avoidance is equally shared between the robots and the robots are expected to take intelligent decisions to avoid one another. We have worked on the Optimal Reciprocal Collision Avoidance Algorithm (ORCA), proposed by

Berg et al. [1], where we have extended the scope of the algorithm by readjusting the assumptions put forward by the authors. One of the main assumptions of the algorithm proposed in [1] is perfect sensing where variations in the measurements obtained by the sensors on a real platform are ignored. However in real world environments, erroneous measurements may be obtained. Therefore in order to implement the algorithm in such circumstances, we are exploring the possibility of introducing a particle filter approach.

Particle filtering in motion planning has been widely used in robotics navigation and tracking. This is mainly due to their well-defined ability to estimate nonlinear dynamic systems. Successful implementations of particle filtering are available in various applications because the approach can fill in the measurements of the system based on estimation and discard erroneous observations thereby making the system stable. One of the main issues faced during the implementation of ORCA is the lack of reliability in velocity computation once global deadlock is achieved. This is important to avoid collision. Also the assumption of perfect sensing makes the algorithm more difficult to implement in a real-life platform. To provide more scope and reliability to the existing algorithm, our intention is to couple particle filtering so that a more consistent solution can be found within the specified time frame. One of the main reasons for using the particle filtering approach is that this provides a likelihood function to estimate the next position without knowledge of the final target. Our objective is therefore to analyze the Reciprocal n-body Collision Avoidance algorithm proposed in [1] and implement a simulation taking a different approach to solve the problem of collision avoidance in mobile robots. This is proposed by appending the particle filtering approach to the existing algorithm. Through this, we hope to prove that this approach will better sample velocities and therefore improve collision avoidance among mobile robots. This approach will also be suited to robots in real-life environments where perfect sensing of velocities between robots cannot be guaranteed.

1.1 Contributions

In this research, significant progress has been made in different areas and will be briefly explained below. The complete description of each task will be available in subsequent chapters.

Understanding the system:

During our literature survey, we exhaustively evaluated various collision avoidance algorithms applied in this area. After this evaluation, we decided to implement the Reciprocal n-Body Collision Avoidance Algorithm [1]. In addition, we also explored several path planning algorithms, topics on linear programming and particle filtering techniques.

Simulation Environment:

Through this research, we have developed a complete simulation package that can be used by other research groups working in the area. The simulation environment that is setup in Matlab consists of no inbuilt functions, thereby making it easier for future users to debug in case of any error. In addition, the simulation environment assigns tasks to the robots, geometrically obtains collision-avoiding velocities and calculates the optimal velocity using half planes. In addition, linear programming applied to rectangular constraints is included in our program. Our simulation package also includes particle-filtering approach. This simulation package is a major contribution to this field of study.

Improvement in the Existing System:

In the existing linear programming approach, the ORCA planes were placed within circular constraints. We were able to show that rectangular constraints can also be used and the results bear this out. In addition, the introduction of particle filters improved the collision detection percentage compared to just applying ORCA on robots. This is a major contribution to this area and can be improved in further studies.

Chapter 2

Related Work

Over the years there have been different methods proposed and researched in the area of path planning and obstacle avoidance in mobile robotics. In this section, we present the related work in the area of path planning, obstacle avoidance and velocity obstacle.

2.1 Global Path Planning

An elaborate discussion on the importance for calculating configuration space of obstacles during global path planning has been provided by Faverjon and Tournassoud [2]. During path planning, it becomes important to predict the obstacle and thus reorient the calculated path corresponding to the target position. Perez [3] gives a complete overview of the various roadmaps, cell decomposition and potential field methods in the area of path planning. The usefulness of that approach lies in the fact that the entire path from the starting position to the target position can be calculated offline. In [4] one of the main assumptions made by the authors is that the entire model of the surrounding feature of the environments is made to be available, which may be unrealistic and not conforming to real-life applications. Also these algorithms are inapt for obstacle avoidance mainly due to a slow and complex procedure for robot motion planning.

2.2 Obstacle Detection and Avoidance

The purpose of obstacle avoidance is to facilitate smooth movement of the robots in crowded environment and thus in order to achieve the same it is important to consider a larger subset of obstacles rather than a small one. The earlier methods proposed were based on using an Octree and using global algorithms to efficiently calculate a path with the estimations of free space. However they had several disadvantages as to being offline or the algorithms becoming exponential once the degrees of freedom increase. A geometric approach has been proposed in [2] so as to calculate the distance between two moving bodies by considering the parallelism, perpendicularity, and concentricity of the body. Furthermore as the problem were being investigated, it became evident through different methods in [5] and [6] that it is vital to consider the bodies as decision making entities so as to utilize the reactive nature of the objects. [7] provides an interesting overview of the various methods that have been used for obstacle avoidance. The illustrated methods are based on edge detection, certainty grids and potential field methods. Edge detection is performed by utilizing the visible edges around the robot or the obstacle. [8], [9], [10] and [11] use edge detection as the main tool or in combination with other techniques for obstacle avoidance. The existence of visible edges was obtained through global path planning and line-fitting algorithms. However, erroneous readings by the sensors mainly due to the faulty directional approaches result in inaccurate locations of the obstacle or of the edges. Another popular approach that is worthy of a mention is that of the Certainty Grid for Obstacle Representation ([12], [13], [14]), wherein a probabilistic method is used for obstacle detection. Using 24 sensors, the robot scans for nearest obstacle and calculates the distance radially from it in an offline mode. Based on the data, adjustments are made to the robot's trajectory towards the target.

The approach of virtual force field (as illustrated in [7]) where a virtual repulsive force is exerted towards the robot is also used for obstacle detection. An algorithm was developed for rapid calculation of all the combined repulsive forces. Nonetheless there were problems that arose while applying the algorithm. One of the several causes was the drastic change in the repulsive force that resulted in oscillation and unstable motion of the respective robots. Also too much data about the obstacle led to the loss of information about the exact location of the obstacle. Hence in order to counter the issues faced in the above method, Vector Field Histogram (VFH) [7] was developed by using a two-stage process, wherein the entire characteristic of the environment is stored in the form of a Cartesian Histogram grid, which is continuously modified by the sensors in real-time. Furthermore, a polar histogram of one dimension is constructed, so as to estimate the transitory location of the robot that would in turn help in mapping the generalized position of the obstacle with respect to a specific sector. Also, the data required for the procedure is stored in the form of reference values of the drive and steer controllers of the vehicle. Subsequently researchers have developed improved versions of the VFH algorithm in the form of VFH+ [15] and VFH*[16]. The VFH+ algorithm is applied through four stages where the first three steps focuses on laying out polar histograms describing the environment of the robot’s current position.

2.3 Velocity Obstacle

Fiorni and Shiller [17] first suggested velocity Obstacle (VO), where a successful approach of dynamic collision was formulated such that a robot would be able to select a velocity that would result in collision avoidance by just knowing the velocity of the other robot. Further research on VO has resulted in a number of applications like multi robot navigation [18], independent motion of robotic wheelchair through any kind of packed environment [19], mobile manipulation of an autonomous robots for services [20], extensively used for air traffic coordination and also to intimate drivers of possible collision ([21], [22]). Another important application of this approach is that it is also used to study the behavior of groups of animals like birds and fishes [23] and also such reactive behavior with respect to this approach can be effectively used in developing simulation and other AI applications [24].

In order to apply VO in real time, it wouldn't be feasible for each robot to communicate every possible data to other robots all the time. The minimum requirement would be to take into account of the reactive nature of the other robot. In this case it would be enough if one robot can communicate just the position and the velocity when both of them are on collision path. This technique is referred to Optimal Reciprocal Collision Avoidance [1]. Similarly Reciprocal Velocity Obstacle (RVO) [25] is another algorithm based on VO for robot-robot collision avoidance in which each robot selects a velocity that lies outside the velocity obstacle of the other robot and an average of its current velocity. This algorithm mainly explored the cycle of sensing and reacting in moving and static obstacles. A neighboring region was developed for each robot with respect to all other obstacles around it in the surrounding environment. The size of the region is directly proportional to the average speed around the obstacles. This approach was tested on differently shaped environments like circle, narrow passages and fast moving obstacles. One of the main results is that there is a linear relationship between the number of robots and each cycle of sensing and reacting. However this approach has been found to be collision-free under certain unique conditions and this can easily fail if both the robot’s choice of velocity lies in the same region. In [26], each robot develops a common velocity obstacle (CVO) map that includes all the robots around it in the surrounding environment. This is eventually shared between all the robots through the proposed CCA (Cooperative Collision Avoidance) method. To have a more defined selection of velocity a parameter known as Reachable Velocity (RV) is used. In order to consistently select a velocity to avoid collision, Reachable Available Velocity (RAV) is used by the CCA method throughout.

Mathematically RAV is defined as

$$RAV = RV \cap \overline{VO} \tag{1}$$

Comparing with the VO method [17] where the velocity of robots remain constant due to which robots collide, the CCA method aims to find collision avoidance velocities for the robots in each sampling time. Furthermore, the CCA method computes a larger scope of RAV, which would find collision avoidance velocities in instances where the VO method fails to do so.

There are further studies on velocity obstacle avoidance but are not listed here due to their relevance with our approach. The active and reactive nature of the initial method led to generation of recursive probabilistic velocity obstacles as in [7] and [27] where the main issue was that it was never consolidated to a specific point. Mostly the methods in [25] and [26] worked well with two robots but were never consistent when more robots were involved in the same dynamic environment. Another main issue found with [1] and [28] was that there were diminished collision cones computed in various scenarios, which eventually led to collisions in the corresponding time windows. In [29] using overhead video camera sensor reading were obtained and thus a hybrid velocity obstacle was computed that resulted in oscillation and collision free movement of robots. Here the ClearPath efficient algorithm is used to spearhead motion for each robot in case no collision is imminent and also depending on the visibility of HVO. In case there isn't any then it is possible that the robots might collide with each other.

The main approach in most of the algorithms using Velocity Obstacle [2] is that there is minimal dependency of robots on each other to avoid any kind of collision. In contrast to considering the robots as wholesome polygonal structures, the first step would naturally be to consider each one as a point of mass and thus describe the movement of the respective body corresponding to the target position. Even after obtaining a set of collision avoidance velocity, we need to have an efficient optimization method to select a velocity closest to the preferred velocity. Using k-means algorithm should help one to find the most nearest velocity from the available set to avoid collision. This can be done using triangle inequality [40], or balltree algorithms [41], which would help find feasible solutions with probability distribution representing different types of data. Another popular approach used effectively is [42] where the cell partition method is used to store the nearest neighbors in an effort to reduce computational power. Finally after studying the nature of the VO algorithm it was decided to go ahead with [38] with the linear programming approach with a modified technique when compared with [1].

2.4 Particle Filter in motion planning

Visual Tracking, Speech recognition and neural information processing ([30], [31], [32], [33]) are some of the applications among the many that use particle filters.

Object Tracking using vision techniques was mainly based on contours and color features of the object. Using particle Filter for such applications (such as [34], [35]) provided a likelihood estimate and thereby better sampling and tracking of objects. Similarly for path planning in mobile robots, particle filter is to estimate the position and is used in a Bayesian Network. Using laser range data similar approach has been tested on multiple robots ([36], [37]). Hence, research over the years has showcased that estimation of next sample with respect to different applications has improved system stability and consistency.

Chapter 3

Velocity Obstacle

Navigating a mobile robot in dynamic environment is a considerably difficult task as there is sufficient amount of planning required with respect to the path and the velocity profile computed. Path planning is defined as calculating a trajectory for the robot from the start to the end position without colliding with any object on its way. Simultaneously a reasonable velocity profile needs to be maintained with respect to each robot so that the robot does not hit another robot while traversing towards its target position. Thereby Fiorni and Shiller [17] came up with a geometric and dynamic approach for robots to avoid collisions. The VO method set the platform for various algorithms for dynamic motion planning due to its advantage of giving a dependable geometric representation for allowing robots to shift and move away from obstacles. However, the method is defined for a specific time frame since in dynamic environments the constraints would keep evolving at different intervals. Hence the solution for each window of time may not be same. The biggest turning point with respect to this method is that the constraints of the robots in consideration is only the position and the velocity that makes it a simplified process and also provides lot of avenues for further exploration for dynamic motion planning especially for a large number of robots.

Velocity Obstacle mainly refers to the set of velocities that would result in collision between robots with in a specific time window. In Figure 3-1there are two robots A and B moving with velocities V_A and V_B at time t_0. A and B are interchanging positions with each other and since their trajectories are pointed in opposite directions along the same line, an imminent collision is expected within time t_1. The velocity profile of A would contain velocities enabling it to collide with B on its way. Hence, Velocity Obstacle is defined as $VO_{A|B}$, which contains velocities of A with respect to B that would result in collision between the two.

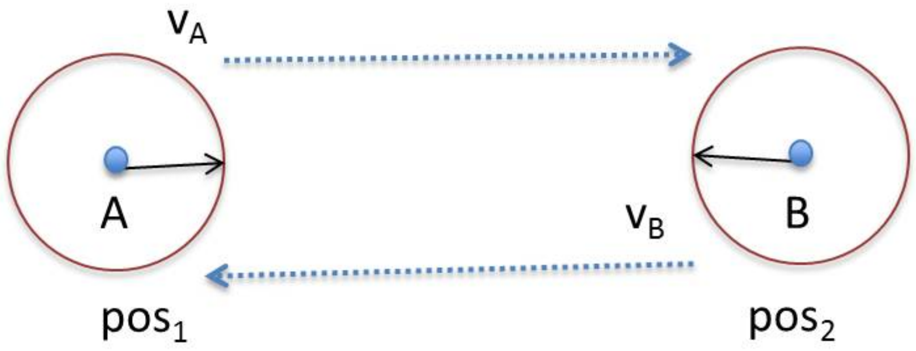


Figure 3-1 Robots A and B are moving with velocities V_A and V_B respectively. A and B are swapping positions. Conclusively they are travelling with velocities that would result in collision between the two robots

Now, VO of robot A with respect to B is used to modify the current velocity of A accordingly so that it can move around B. Initially the representation of A is reduced to a point mass and B is illustrated with the radius of A in the form of a circle. This approach to depict what is known as Velocity Space. Before moving any further one would have to consider relative velocity between A and B, which is,

$$V_{A/B} = V_A - V_B \tag{2}$$

Suppose the line λ_{AB} corresponding to the relative velocity $V_{A/B}$ goes through B, then

$$\lambda_{AB} \cap \bar{B} \neq \emptyset \tag{3}$$

Geometrically VO for robot A with respect to B is defined in the form of a Collision Cone (Figure 3-2), illustrating the colliding velocities of robot A and B. Collision Cone (CC_{AB}) can be mathematically represented as:

$$CC_{AB} = \{V_{AB} | \lambda_{AB} \cap \bar{B} \neq \emptyset\} \tag{4}$$

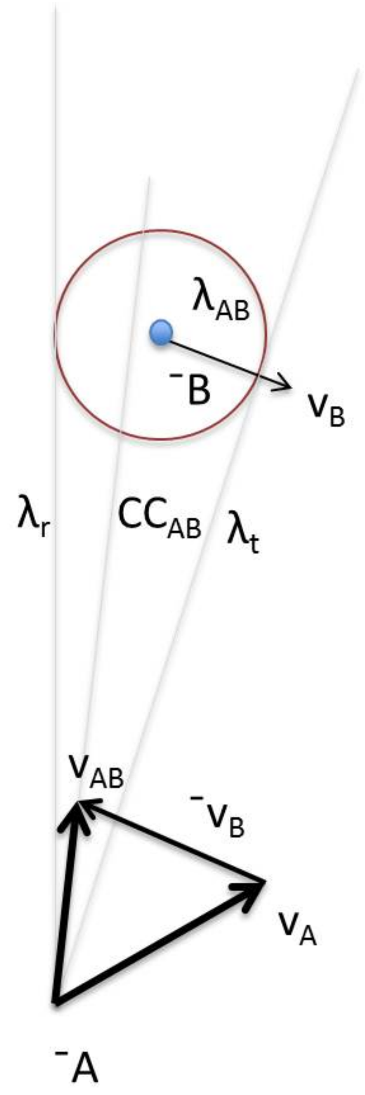


Figure 3-2 Geometric representation of Relative Velocity V_{AB} and collision cone CC_{AB}

In the figure above (Figure 3-2), $V_{A/B}$ is the relative velocity between A and B while λ_{AB} is the tangent line to the relative velocity $V_{A/B}$ intersecting B. This would indicate that a collision is expected to occur in the next time window t_1 if the respective robots are moving further in the same velocity. As it is clear, any velocity selected from an external area with respect to CC_{AB} would help in avoiding any obstacle coming in the way of the robot and would ensure collision avoidance within the specified time frame. Another angle of observation is that the CC_{AB} also divides the velocities into colliding and collision avoiding. As CC_{AB} is elaborated for a definite obstacle, then in case of multiple obstacles by simply adding the velocities of other robots and thereby translating the CC_{AB} would develop Velocity Obstacle (refer Figure 3-3 and Figure 3-4). An absolute equivalent partition of obstacles would thus help to compute the velocities for A that would avoid B but also ensure that A is back on its computed path.

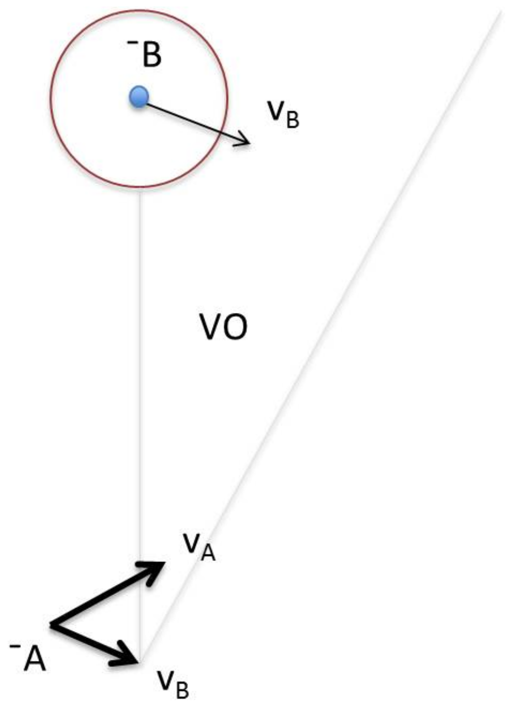


Figure 3-3 VO is constructed by adding the velocities of B to the set CCAB. The addition is in the form of Minkowski sum

Therefore,

$$VO = \; CC_{AB} \; \Theta \; V_B \tag{5}$$

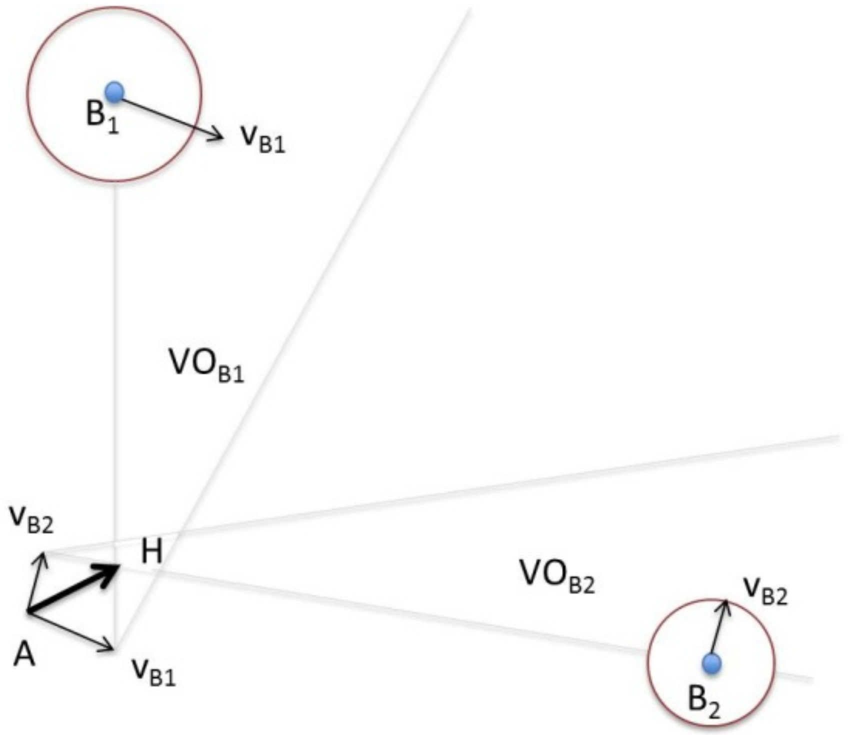


Figure 3-4 The collision cones and velocity obstacles of B_1 and B_2 that is computed with respect to A. Combining the VO for both would result in a set of velocities that allow A to avoid both B_1 and B_2. The VO is recomputed for a different time interval.

Now to avoid more than one, for instance n-obstacles

$$VO = \cup_{i=1}^{n} VO_i \tag{6}$$

Now selection of velocities for robot A outside of VO would not result in collision with B whereas if V_A lies on the boundary of VO would result in a very close movement along the side of B which might eventually lead to collision. There are mainly two assumptions made here in developing VO by Fiorni and Shiller [17]. The robots are considered to be circular in shape so that the configuration space is reduced and thereby result in easier computation of VO. Also for each specific time frame the robots are expected to maintain the same velocity throughout that would allow a smaller area for RAV.

Chapter 4

Optimal Reciprocal Collision Avoidance Algorithm

4.1 Overview

Multi-robots provide fascinating areas of research like assigning specific tasks and ensuring those are carried out by each of them in perfect synchronization. As opposed to assigning a complex task to a single robot, it was found easier to breakdown the load and assign the same to a number of homogenous robots. Such an area has gained wide prominence for various applications like space, underwater and ecological exploration, and entertainments, industrial and similar systems where they can be efficiently utilized. In such an instance, the fundamental requirement is to make sure that all the robots are well coordinated at all times. One of the simple tasks would be to have them moving around in a common environment without coming on the path of each other. Therefore, the approach of Reciprocal n-body Collision Avoidance Approach (RCAA) is a way by which multiple homogenous robots are made to move around so that they hinder in each other's paths. A decentralized concept is considered so that each of the robots can take care of itself and accordingly the system would be stable for the expected period of time.

Reciprocal n-body collision avoidance is another view of collision avoidance where each of the robots is assumed to be decision-making entities. This mode of operation of collision avoidance is mainly a heavy-handed method since each robot has to continuously execute a set of techniques to avoid each other in the specified time window. Since each of the robots has to decide its next course of action within a stipulated time, it has to perpetually carry out a cycle of sensing and reacting.

In order to find a consistent solution for such an instance where we have a collection of robots with the same behavior, there has to be a set of parameters set. A major measure is to use a holonomic robot with simplified characteristics, which would permit the robot to navigate in any direction. Since a continuous cycle of sensing has to be sustained, each robot is assumed to have perfect sensing nature. Furthermore all the robots are expected to have the same, size and shape. Since this is such an exhausting approach, a 2-dimensional velocity vector describes the maneuver of each robot. The importance of using a holonomic robot is mainly so that the navigation of robots can easily be controlled in any kind of congested environment. This is because the robot has the flexibility to turn in any direction without any kind of change in its pivot point. Another advantage in using such a robot is that it would also allow the robot to change direction from any type of complex juncture. Moreover such kind of robots would just need simple hardware that can easily be intercepted with preprogrammed instructions with adaptability of dynamic processing rather than on the basis of being statically defined.

This approach also has another angle to it where the algorithm ensures that the robots avoid collision for a predetermined period of time. The window for this period can be defined based on the total number of robots or the efficiency of the instructions programmed. The optimal velocity that each robot selects is from the corresponding velocity space where a favorable region is found using linear programming.

4.2 Initial Assumptions

In order to proceed with the problem as outlined above, the components being used need to be less complex as the solution itself is expected to use up a lot of resources. The environment in consideration here is a simple 2D space with circular robots of the same radius. As elaborated in [1], we would need to set parameters that would define state of the robots. The state of the robots refers to the position and velocity of each robot. As discussed before, each and every robot is assumed to be able to obtain the velocity of other robots. The current position of a robot known as A is defined as POS_A with current velocity as VEL_A. Along with the current state of the robot, other values that are set are $VEL^{max}{}_A$ and $VEL^{pref}{}_A$. The maximum velocity for each robot has to be fixed so as to given an upper bound in order to avoid permanent looping. Similarly a preferable velocity has to be also set so that the robot by itself should move around in optimal velocity when no obstacle is in its way. Furthermore, the collision avoidance of robots is set for particular window of time known as τ.

An overview of the algorithm indicates that to guarantee non-aligned collision avoidance among the robots it is imperative of each and every robot to avoid coming in the way of other. Another important point to be considered is that other than knowing the velocities and positions of each other there are no other means of communication between them. As there in no centralized coordination system in the environment, the aim of the robots should be to move around in a velocity close to the $VEL^{pref}{}_A$ so that consistency can be maintained so that a thorough consistency can be maintained all through the process.

4.3 Reciprocal Collision Avoidance

Before elaborating further on collision avoidance on n robots, it is important to further define and elucidate velocity obstacle of robot A with respect to robot B considering the shape of the robots as well as other constraints ([1], [17]).

A collection of relative velocities of robot A with respect to B, thereby resulting in collision within the time τ is known as velocity obstacle $\mathrm{VelOb}^{\tau}{}_{A|B}$. All the robots are assumed to be in homogeneous with disc shape of fixed radius r. The characteristics robots are mathematically formulated as D(POS,r) where POS and r refers to position and radius respectively. The position of each robot, for instance robot A is POS_A assuming with center of the open disc at POS_A.

$$D(POS, r) = \{q \mid ||q - POS|| < r\} \quad (7)$$

then,

$$VelOb^{\tau}_{A|B} = \{VEL \mid \exists\, t\, \mathcal{E}\, [0, \tau] :: t\, VEL \in D\,(POS_B - POS_A, r_A + r_B)\} \quad (8)$$

In equation (8) the velocity obstacle of A with respect to B, $\mathrm{VelOb}^{\tau}{}_{A|B}$ is defined such that all velocities, VEL within the time window t would result in collision between the robots A and B at positions POS_A and POS_B (refer Figure 4-1).

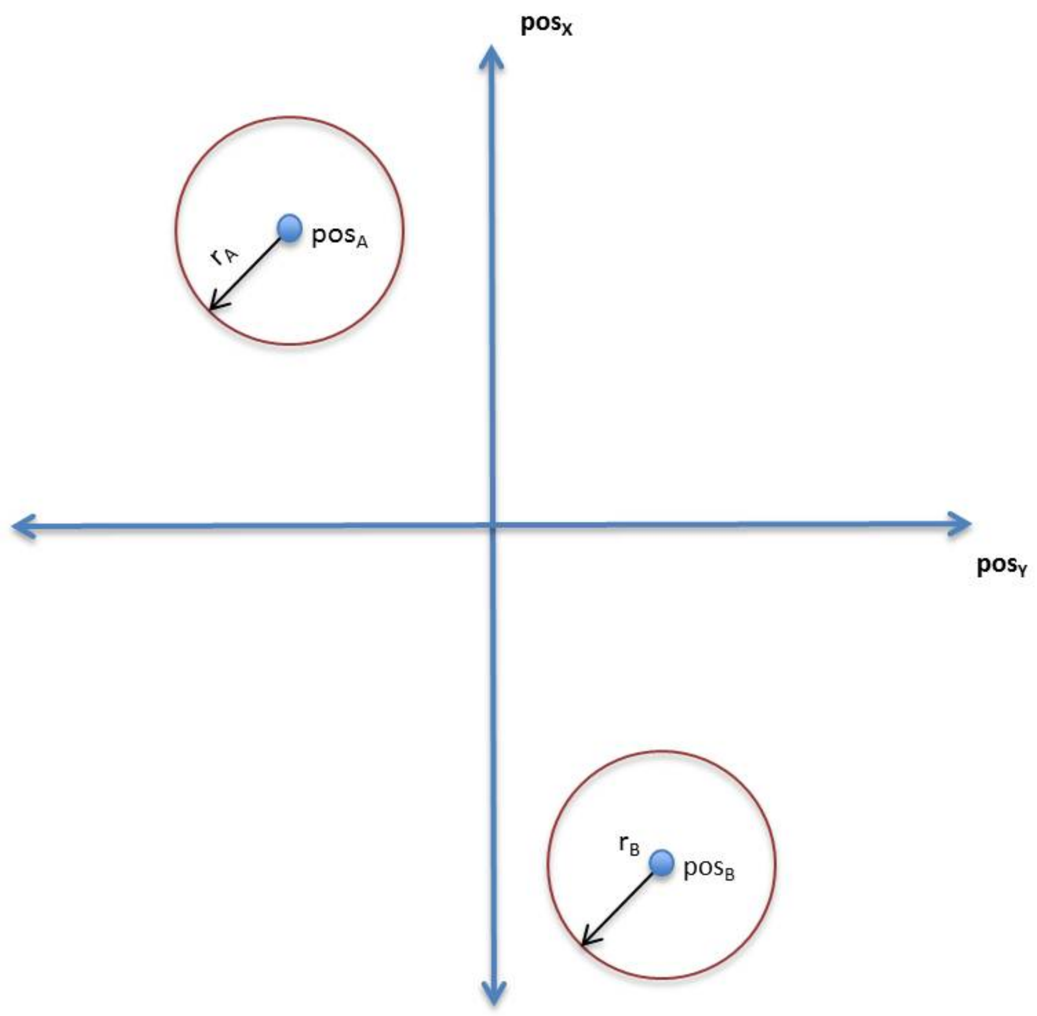


Figure 4-1 Placement of robots A and B with position POS_A and POS_B. Here the robots are place in a Cartesian coordinate frame so as to depict as how the positions are obtained, that is each of the coordinates are collected based on the positions of the centers of each robot.

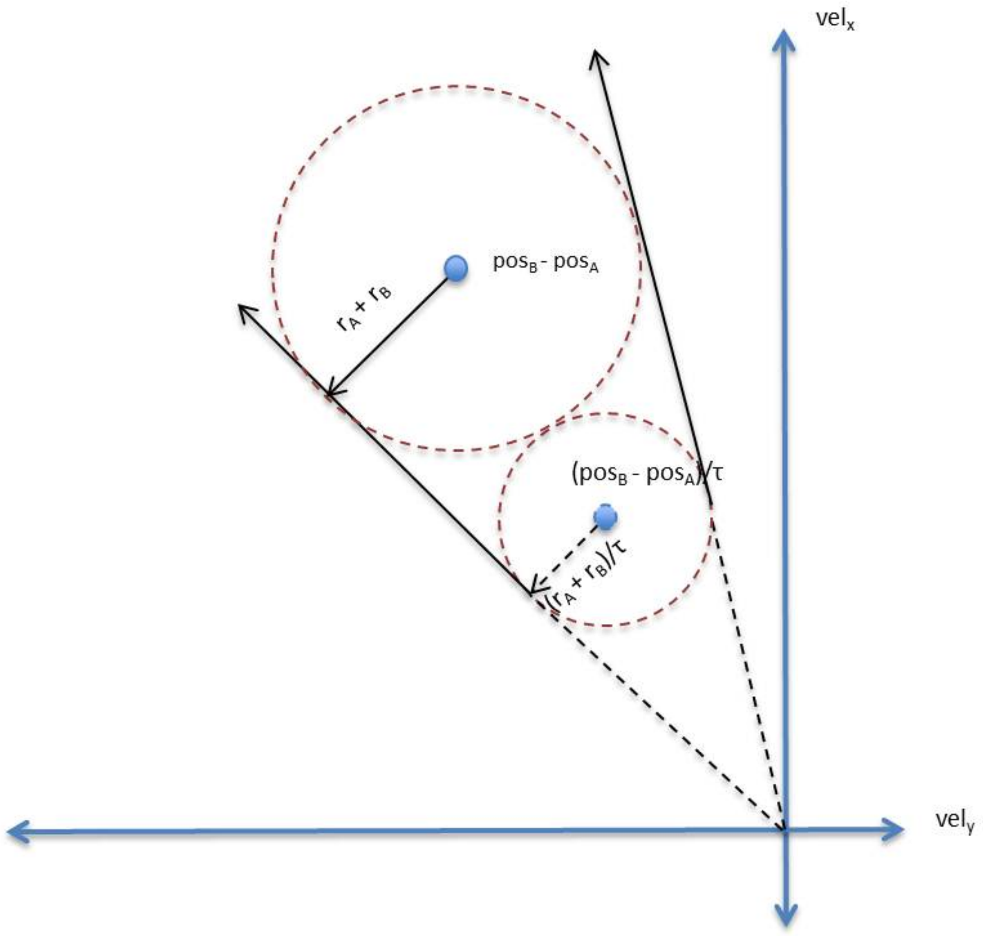


Figure 4-2 Construction of VelOb$^{\tau}{}_{A|B}$ of robots A and B with position POS$_A$ and POS$_B$.The geometric illustration is in the form of a cone from the origin with the each of the two circles having theirs radii computed as $r_A + r_B$ and $(r_A + r_B)/\tau$. The centers are also similarly computed as POS$_B$ – POS$_A$ and (POS$_B$ – POS$_A$)/τ , where τ is the specified time frame. The area enclosing the two circles with the respective tangents is the defined velocity obstacle.

Before we delve deep into the mathematical and geometric illustration of Velocity Obstacle it is important to go through the Minkowski sum. It plays a major part for applications related to motion planning, object containment, collision avoidance with respect to both 2D and 3D Computer Graphics. In simple terms, Minkowski Sum is the addition of two sets of position vectors in Euclidean space.

i.e., $\{\ a + b : a \in A, b \in B$ (9)

Considering robots A and B, their respective current velocities are depicted as vel_A and VEL_B where $VEL_A - vel_B \in VelOb^{\tau}{}_{A|B}$ and similarly $VEL_B - VEL_A \in VelOb^{\tau}{}_{B|A}$, thus indicating that A and B with collide with each other if they keep navigating in the same velocities (refer Figure 4-2). On the contrary if $VEL_A - VEL_B \notin VelOb^{\tau}{}_{A|B}$ and $VEL_B - VEL_A \notin VelOb^{\tau}{}_{B|A}$, thus ensuring that during the defined time period τ, they would not come in the way of each other. As the velocity obstacle is represented in the Figure 4-2, we see that they are symmetric in the origin.

In order to define and compute the collision avoiding velocities, Minkowski sum (refer Figure 4-3) is put into effect where for any $VEL_B \in VEL_B$ and $VEL_A \notin VelOb^{\tau}{}_{A|B} \oplus VEL_B$. This would permit the robots A and B to be collision free for the definite time window τ. Thereby in order to further set forth the set of collision avoiding velocities, $ColAv^{\tau}{}_{A|B}$,

$$ColAv^{\tau}{}_{A|B}(Vel_B) = \{ VEL | VEL \notin VelOb^{\tau}{}_{A|B} \oplus VEL_B \} \tag{10}$$

Such a collection of sets VEL_A and VEL_B is known as reciprocally collision avoiding where $VEL_A \subseteq ColAv^{\tau}{}_{A|B}(VEL_B)$ and $VEL_B \subseteq ColAv^{\tau}{}_{B|A}(VEL_A)$. There are also cases where $VEL_A = ColAv^{\tau}{}_{A|B}(VEL_B)$ and $Vel_B = ColAv^{\tau}{}_{B|A}(VEL_A)$, which are thereby known as reciprocally maximal.

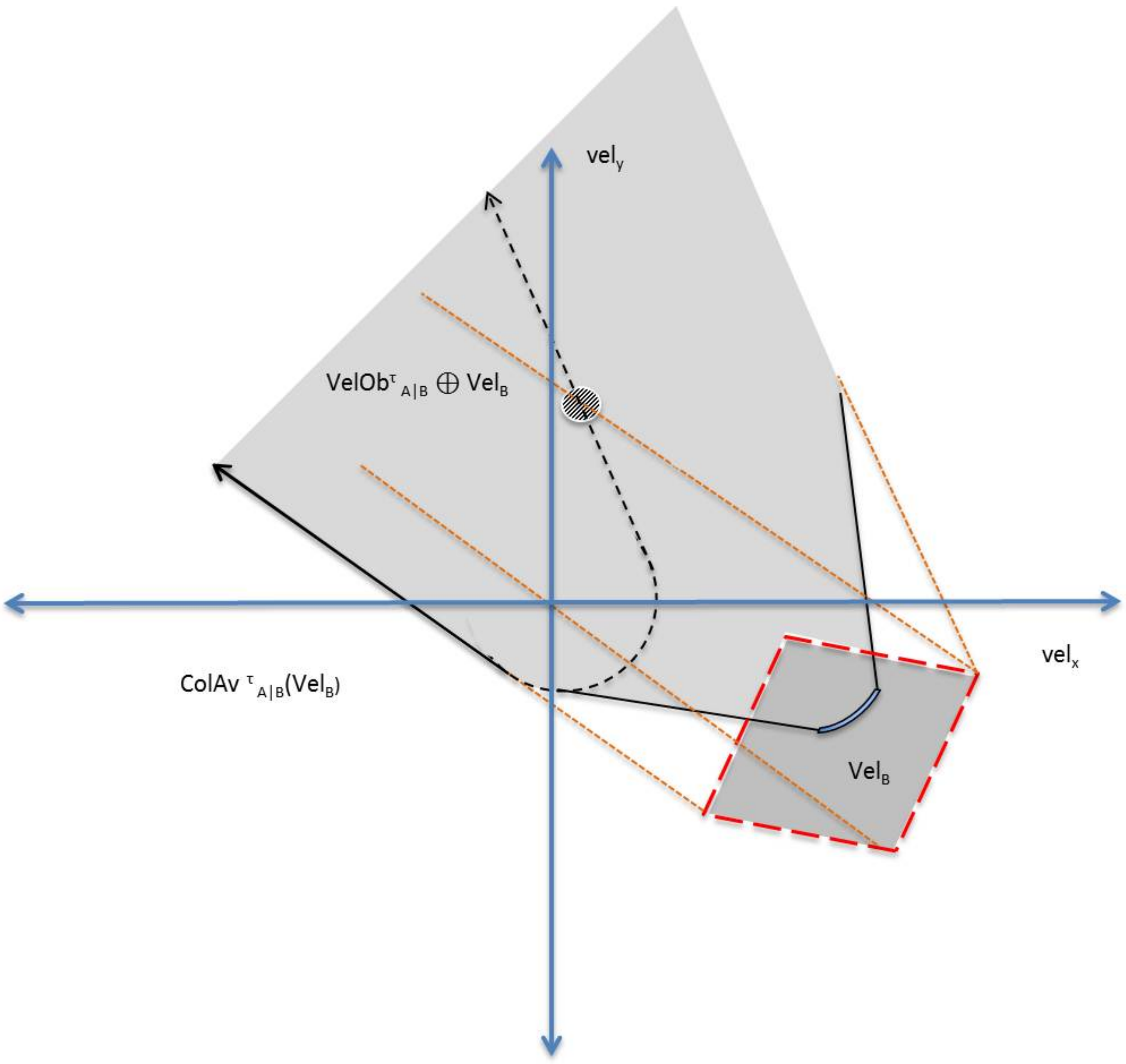

vel$_y$
VelOb$^\tau$$_{A|B}$ $\oplus$ Vel$_B$
vel$_x$
ColAv $^\tau$$_{A|B}$(Vel$_B$)
Vel$_B$

Figure 4-3 As Minkowski Sum is addition of two sets of positive vectors in Euclidean space, of VO and VEL$_B$. Here ColAv $^\tau$ $_{A|B}$(VEL$_B$) is the set of collision avoiding velocities. The area shown in the form of a parallelogram in dark shading with red borders, VEL$_B$ is the complement of the Minkowski sum of VO and VEL$_B$. ColAv $^\tau$ $_{A|B}$(VEL$_B$) is formulated on the assumption that B would choose velocity from VEL$_B$.

As there are reciprocally maximal velocities and preferred velocities for A and B, it would be ideal to actually choose a velocity that is optimal using linear programming. This is because there can be boundless sets of VEL$_A$ and VEL$_B$ that would satisfy the needful as stated above. Due to the same reason it's important to select a velocity that is near to the set of optimization velocities but at the same time fully utilizing the set of velocities allowed. As the problem is being studied in a two dimensional space, |VEL| denote the measure of the entire set of velocities in area R2.

To proceed ahead we would need to define the Optimal Reciprocal Collision Avoidance for a robot A coming in the way of B or vice versa. Let it be known as $ORCA^{\tau}_{A|B}$ for A with respect to B and similarly $ORCA^{\tau}_{B|A}$.As the velocities for such a set has to be reciprocally collision avoiding and maximal since, $ColAv^{\tau}_{A|B} = ORCA^{\tau}_{A|B}$ and $ColAv^{\tau}_{B|A} = ORCA^{\tau}_{B|A}$. The other important measure to be in consideration is that the rest of all other pairs of sets of reciprocally collision avoiding velocities VEL_A and VEL_B, $VEL_A \subseteq ColAv^{\tau}_{A|B}(VEL_B)$ and $VEL_B \subseteq ColAv^{\tau}_{B|A}(VEL_A)$.. This is also applicable only when the radii of all the robots r >0.

Thus,

$$|ORCA^{\tau}_{A|B} \cap D(VEL^{opt}_{A}, r)| =| ORCA^{\tau}_{B|A} \cap D(VEL^{opt}_{B}, r)| >= \min(|VEL_A \cap D(VEL^{opt}_{A}, r)|, \min(|VEL_B \cap D(VEL^{opt}_{B}, r)|). \quad (11)$$

Therefore it can be deduced that the set $ORCA^{\tau}_{A|B}$ and $ORCA^{\tau}_{B|A}$ consists of higher velocities that are near to VEL^{opt}_{A} and VEL^{opt}_{B} than the rest of the number of reciprocally collision avoiding velocities. Another important part of this strategy as discussed above is that the responsibility to avoid collisions is shared equally between the robots. Therefore the distribution of velocities permissible to the defined set of conditions that is close to the optimal velocities is the same for A and B.

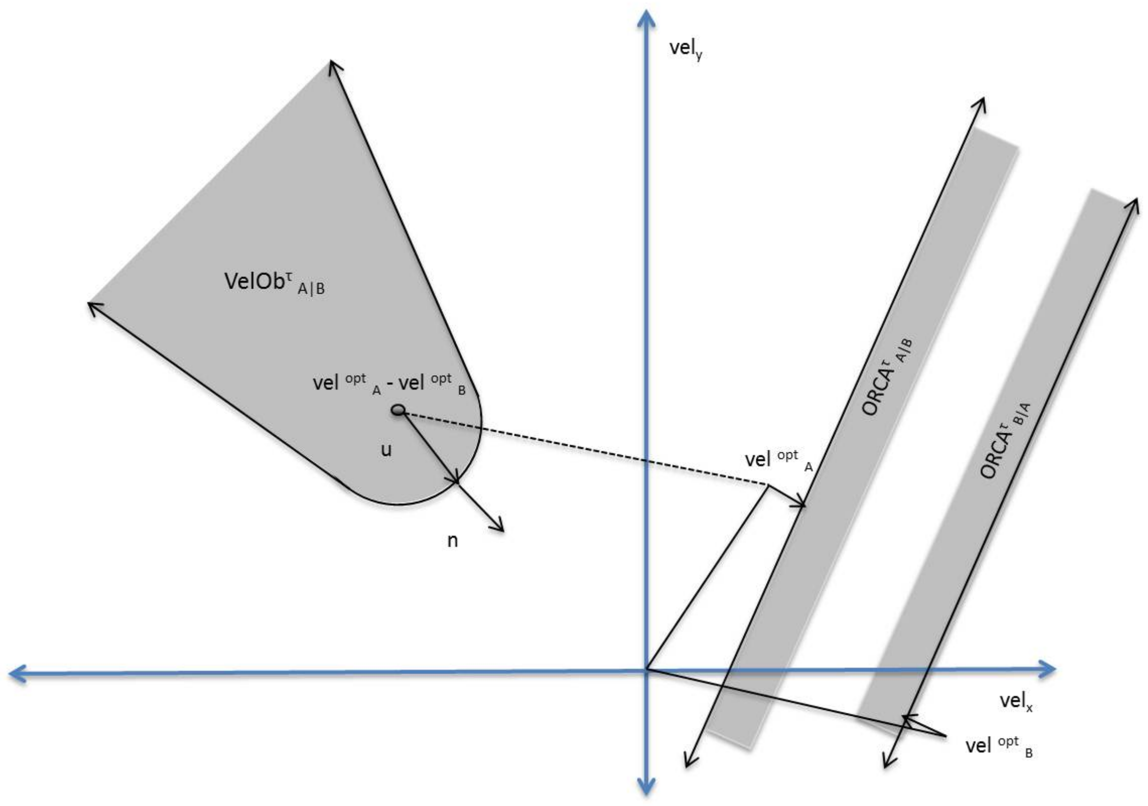


Figure 4-4 Geometric depiction of ORCA$^{\tau}{}_{A|B}$ and ORCA$^{\tau}{}_{B|A}$. Initially the velocity obstacle VelOb$^{\tau}{}_{A|B}$ is calculated for A with respect to B's current position. Also, ORCA$^{\tau}{}_{A|B}$ is composed of those velocities that would enable A to avoid colliding with B in time τ , which is calculated using the reciprocal collision avoidance method. Here u is the vector that is placed at shortest distance from VEL$^{opt}{}_{A}$ – VEL$^{opt}{}_{B}$ to end of the cone depicting VelOb$^{\tau}{}_{A|B}$.

From the geometric description of ORCA$^{\tau}{}_{A|B}$ and ORCA$^{\tau}{}_{B|A}$, we see that VelOb$^{\tau}{}_{A|B}$ is symmetric towards origin. Suppose the velocities with which A and B are moving around is in VEL$^{opt}{}_{A}$ and VEL$^{opt}{}_{B}$ and that they are going to collide in time τ. This can be represented as

$$\mathrm{VEL}^{opt}{}_{A} - \mathrm{VEL}^{opt}{}_{B} \in \mathrm{VelOb}^{\tau}{}_{A|B} , \tag{12}$$

Assume that u be the vector from VEL$^{opt}{}_{A}$ – VEL$^{opt}{}_{B}$ to the nearest point on the bounding area of VelOb$^{\tau}{}_{A|B}$,then

$u = (\text{argmin}_{VEL \in \partial VelOb\tau\, A|B} ||VEL - (VEL^{opt}{}_{A} - VEL^{opt}{}_{B}) \text{-} (VEL^{opt}{}_{A} - VEL^{opt}{}_{B})$

(13)

Now, if n is the outward normal of the boundary of $VelOb^{\tau}{}_{A|B}$ at point $(VEL^{opt}{}_{A} - VEL^{opt}{}_{B}) + u$. In order to avoid collision within the given time window, τ then u is the minute adjustment applied to relative velocities of A and B so that they would avoid colliding with each other. As the responsibility is equally shared between the two to avert collision, A alters it's by velocity by (1/2) u and also expects B to take care of its velocity the same way. From the figure, $ORCA^{\tau}{}_{A|B}$ is the obtained half plane that begins at point $VEL^{opt}{}_{A} + ½ u$ that is in the direction of n.

Mathematically representing,

$$ORCA^{\tau}{}_{A|B} = \{VEL \mid (VEL\text{-}(VEL^{opt}{}_{A} + ½ u)).n >= 0\}, \qquad (14)$$

If A and B are not going to collide with each other, that is if

$VEL^{opt}{}_{A} - VEL^{opt}{}_{B} \notin VelOb^{\tau}{}_{A|B}$, even then the optimized velocities selected would still enable them to remain on a collision free trajectory.

4.4 n-Body Reciprocal Collision Avoidance

Moving further, the objective is to obtain a decentralized approach for multiple robots to avoid colliding with each in the same environment. In order to achieve the same, each of them preforms a continuous cycle of sensing and reacting so that it would be able to successfully reach the target point. As is the case of two robots, the set of permissible velocities is obtained for A with respect to B is $ORCA^{\tau}{}_{A|B}$. On the basis of this inference, then the intersection of half planes of n robots contains the set of allowed velocities for A to attain a collision free trajectory with respect to n robots around. This concept can be further represented as,

$$ORCA^{\tau}{}_{A} = D(0, VEL^{max}{}_{A}) \cap \cap_{B \neq A} ORCA^{\tau}{}_{A|B} \qquad (15)$$

Another important observation from the above representation is that the maximum velocity allowed also comes into play. This constraint is to ensure that when linear programming is applied, any kind of unacceptable values for velocity is not used for the selection of optimal velocity. Now, $VEL^{pref}{}_{A}$ is the preferred velocity defined for A and $VEL_{A}{}^{new}$ has to be selected in such that it lies within the region of permitted velocities.

$$VEL^{new}{}_{A} = \operatorname{argmin}_{VEL \in ORCA\tau\ A|B} \ || VEL - VEL^{pref}{}_{A}|| \qquad (16)$$

Now, the robot has to move to the next point with new obtained velocity, thereby in order to calculate the same,

$$POS_{A}{}^{new} = POS_{A} + VEL^{new}{}_{A}\, \Delta t, \qquad (17)$$

As observed in certain scenarios, the robots would be headed in different directions. This presents a challenge to find the right set of optimal velocities that would allow the respective robot to avoid colliding with all of them but also reaches its target position without deviating from the same.

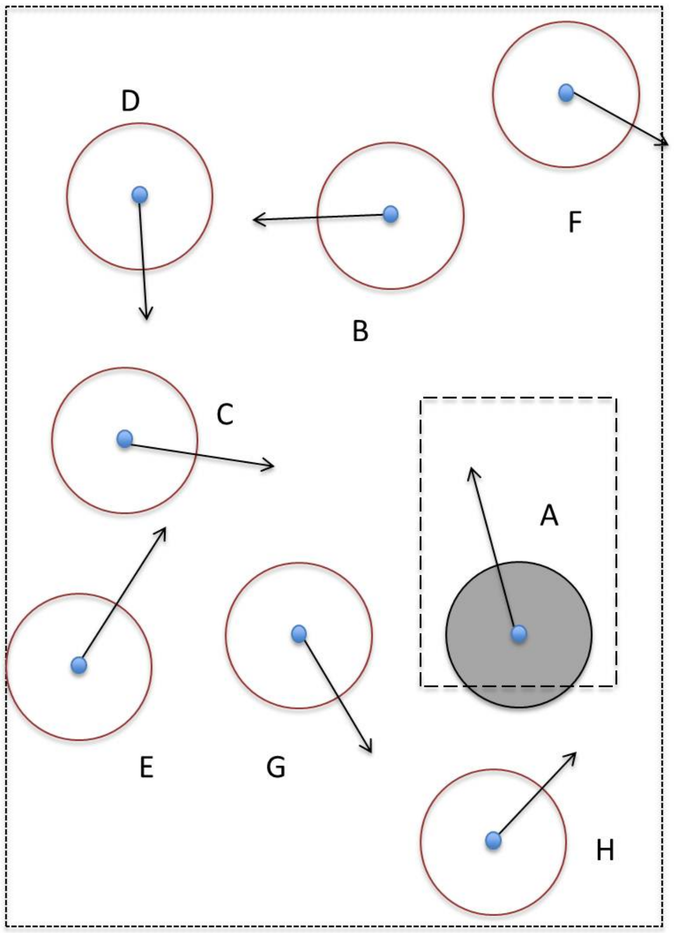


Figure 4-5 Robots around A are moving in different directions. Here the aim is to find a unique path to the target position without bumping into the rest of the robots around. In order to find the set of optimal velocities, we would have to combine the velocity obstacle of A with respect to every other robot.

Thus for the robot to continuously calculate the optimal velocities, a consistent optimization approach is required. As we move ahead, in order to carry out such an exhaustive approach, linear programming is applied to obtain the ORCA region of a robot with respect to the rest of robots, which is elaborated in the subsequent chapter.

Chapter 5

Optimization Algorithm

The main reason to use an optimization algorithm after obtaining ORCA lines of each robot corresponding to the rest is to choose the optimal velocity so that the robots can avoid each other. The set ORCA contains velocities that would allow each one to move away from each other. As there are many ways for a robot to move around other obstacles in, it can be frustrating to find the most optimal route. In many applications like mechanics and engineering, economics, mathematical and management science, there always arises the need to find the most desirable solution iteratively. In such scenarios, there would be a maximizing or minimizing optimization function that would enable us to find the best alternative from the continuously obtained solutions.

In a multi-robot system, there is continuous redundancy and cooperativeness expected throughout so that each of them fulfills their respective duties. Since the collision avoidance velocities are obtained throughout in the form of linear constraints, the linear programming method in [38] was found to be appropriate to the requirements of our system. As the robots continuously obtain velocity and position of other robots, using the RCAA method constraints would be updated for each time frame to ensure collision avoidance. Even though a number of alternate velocities are computed through the ORCA set, an optimal would be to find the velocity that is closest to the preferred velocity but at the same time doesn’t come in the colliding path of any other robot around the respective one.

The optimization algorithm that is used for the selection of velocities for each robot that is on its collision path with the nearest robot around is the Polygonal Sweep

Algorithm [38]. An efficient algorithm was required for working out an optimized velocity so that when each robot tries to adopt collision free velocity; there is systematic efficiency throughout the time frame. Polygonal Sweep Algorithm is actually a modified version of the Line Sweep Algorithm that is applied on a set of lines to check the number of intersection points of the lines with respect to each other. Similarly the polygonal sweep algorithm is applied to check the intersection of any two planes. The obtained ORCA lines are envisioned as planes with feasible region drawn in the respective direction using the ORCA approach. In the optimization approach, the intersecting planes are checked for their feasible region and thus a common region is acquired. The obtained common polygon is used for further processing and the rest of the insignificant region is discarded.

As discussed above, it is evident that there would be a finite number of ORCA lines that would be generated for every robot with respect to another robot. In such a circumstance, it is important that we have a cohering linear programming solution that would help us in finding a reliable solution with respect to a collection of line segments. Hence we approached the problem by considering each line segment to be a linear constraint. Initially a boundary was set that would help us converge to a solution at the earliest. Thus, considering all the logistics that were being used, an R2 plane was initialized with preset values. The objective here was to obtain a favorable region within a desired range. As the ORCA lines are continuously generated within each time window, it is important to enclose them on a common workspace so that a desirable solution is found all the times. For the same reason a rectangle was constructed with the constraints of the robot as dimensions in order to find the feasible region.

For example,

$$P1= [0,0,0,100,100,100,100,0]; \tag{18}$$

where an array is used to store the enclosing boundary values.

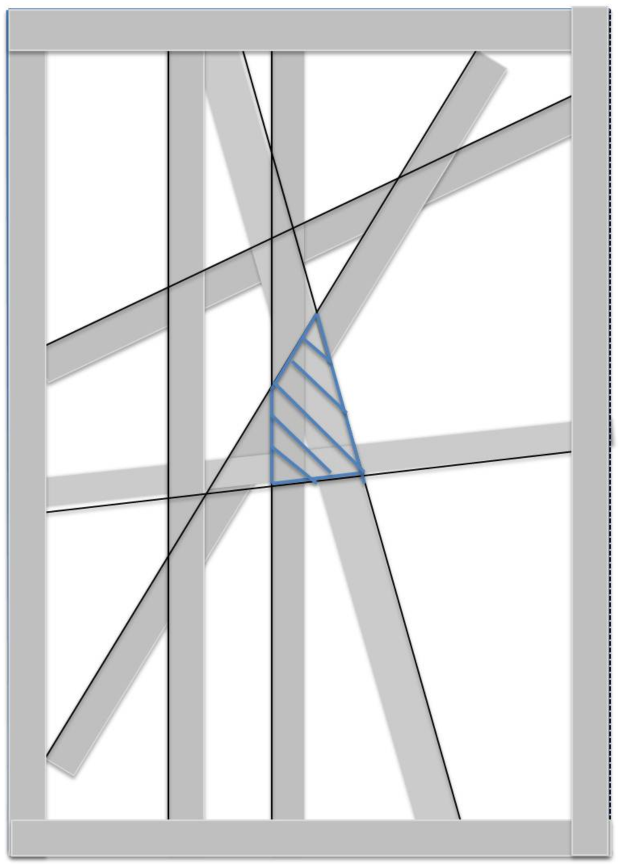


Figure 5-1 ORCA Lines for a robot are generated consecutively as and when other robots are on its path. Hence in order to find the most desirable velocity to avoid the multiple robots on its way, one would have to find the feasible region that satisfies all the constraints as shown in the figure. In the figure we see the multiple ORCA lines enclosed within the defined boundary as defined by the optimization algorithm and also the favorable region in between. The objective would be to find the minimal point that is close to the preferred velocity of the robot so that it does not deviate completely from its path and eventually reaches its target position.

Now let

$$L= \{ l_1,l_2.......l_n \} \tag{19}$$

Here L be the set of linear constraints, that could be represented in the form of

$$a_ix + b_iy <= c_i \tag{20}$$

Here a_i,b_i,c_i are constants and at least one of a_i and b_i is non-zero. The purpose here is to render each line segment as a constraint and also that it will just intersect the set plane in two distinct points such that in the process, we geometrically obtain a probable region which would contain the collection of points:

$$(x,y) \in R^2 \tag{21}$$

The respective aggregations of points are those that would satisfy the n constraints. In order to obtain the set of favorable points, it is obvious that once all the constraints are obtained with respect to each robot one would need to obtain the feasible region first. As there would be a number of line segments intersecting the preset boundary, it is important that we obtain the smallest common region with respect to the intersection points of the lines with respect to the initial set plane. Furthermore the prior step to finding the feasible region would be to obtain all the intersection points of all the constraints. As there would be a considerable number of intersecting points, we required an algorithm that would take as much of less computation time. Thus considering the large number of iteration and steps that were required in the procedure the Line Sweep Algorithm was studied in depth.

5.1 Plane Sweep Algorithm

Computation of intersection of line segment with respect to other line segments holds a lot of practical applications. As discussed in [38] layout of networks, railroad, rivers and roads are stored as curves, which can be subsequently approximated as line segments. The problem that is in consideration here is to compute the intersection point of one line segment with respect to all other line segments that are present in the other group. This is mainly because the line segments represent the orientation of velocity of one robot with respect to all other robots that are present. Therefore with the objective to find a favorable velocity that would avoid collision with each other, linear programming is applied to find the favorable set of velocities.

Using the traditional approach to obtain intersection point of one segment with another would require $O(n^2)$ time. In the current scenario the demand is for an algorithm that is continuously iterative and also at the time consumes minimal resources. Also, the output is dependent not only on the number of line segments but also on the number of intersection points that are acquired in the process. Such algorithms are output sensitive or intersection sensitive since the running time is dependent on the size of output, which is the number of intersection points. Observing the nature of the lines segments one can infer if they would intersect or not. For instance, line segments that are parallel would not intersect at all, however some of them would be coincident on each other. In such cases, each and every point on the line segment can be considered as intersection point. Therefore the following algorithm has been formulated similar to [38].

Consider

$$S = \{ S_1, S_2, S_3, S_4 \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots , S_n\} \qquad (22)$$

to be the collection of line segments, for which the number of intersection points are to be obtained.

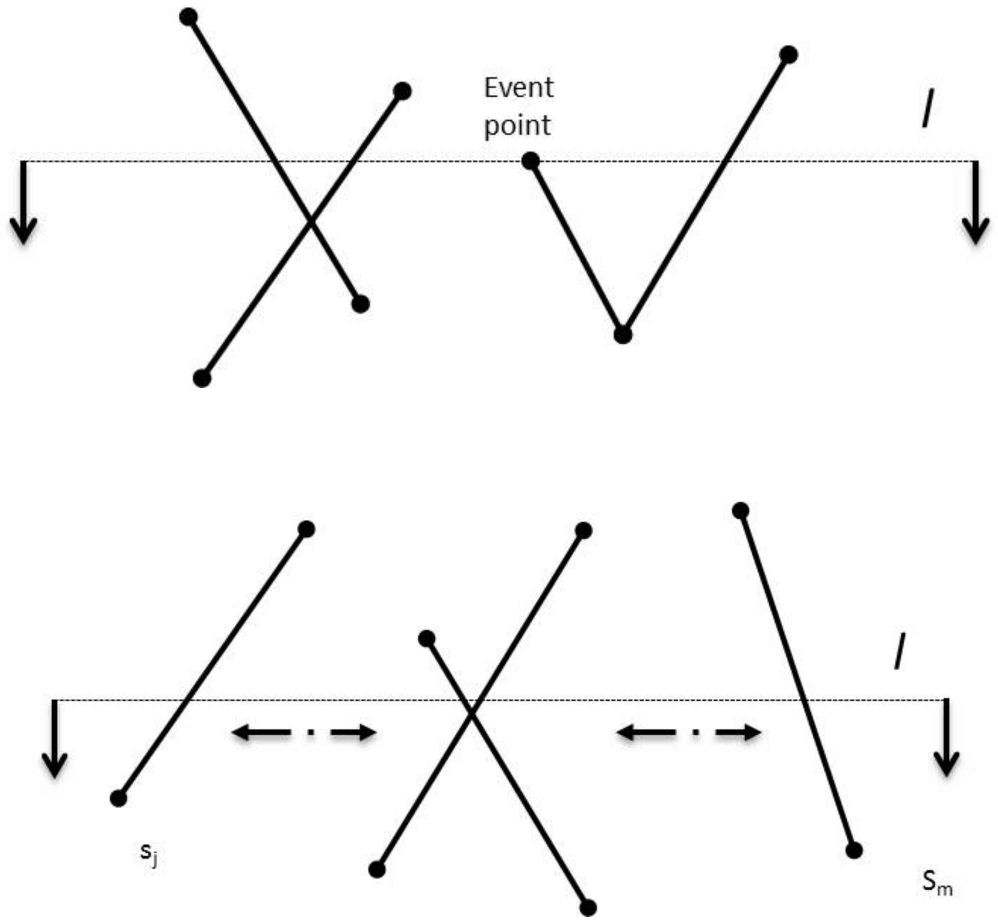


Figure 5-2 In the figure there are multiple sets of intersection of line segments. In order to find all the intersecting segments points, a line *l* is moved across the line segments from one end point to another. Each intersection point is referred to as event point. Line Sweep Algorithm is used here to detect all the event points in the given set.

Consider an imaginary line segment *l* that is sweeping through the entire set of segments known as the sweep line. Here the nature of line segments with respect to each other need to be also considered while applying the optimization algorithm. The movement of the line segment is stored in a queue known as the event queue. The approach here is to deliberate on the orthogonal projection of y axis of either line segments. If at some point, the projections are found to intersect then one can say that the line segments do intersect at some point. As the sweep line is moved through the entire plane of line segments, the algorithm therefore came to be known as the *Plane Sweep Algorithm.*

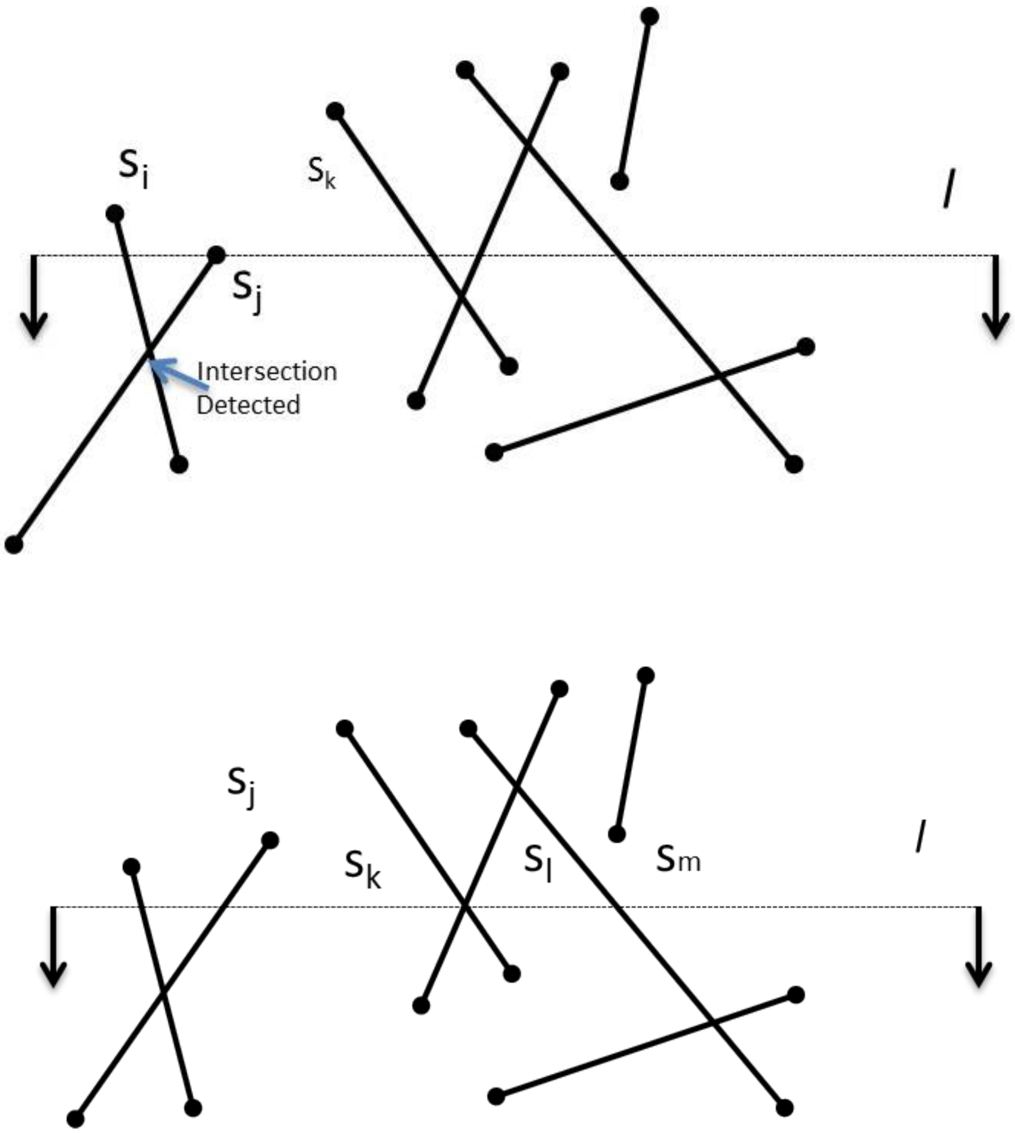


Figure 5-3 Here there are a number of intersection points in the provided scenario. The objective here is to find the order of intersection points among different groups of intersection points using the sweep line *l*. In the specific plane as indicated, it is vital to find every intersection point and record them in the queue in the right order.

As we go into the detail of the algorithm, the first step would be the starting point of the sweep line. As the sweep line encounters each and every, event it is important that each one of them is recorded and thereby the status is updated. As the sweep line start moving downward, the upper endpoint would be an event, as the adjoining neighbors need to be tested (refer Figure 5-3). Several actions are required to modify the placement of the line segments and thus save the intersection points. Another interesting observation here is that after modification it is possible that the nature of the line segment with respect to each may change and thus there might be a probability that two line segments, which may have been far away from one another, may be now lying adjacent to each other. Thus the algorithm is iterative with respect to the fact that until the last event point is sustained upon, each and every occurrence needs to be considered.

The pseudo code for the algorithm is given below:

```
INPUT: S //set of line segments
FOR each event E
    IF E is not empty
        PT=ComputeEvent (E)
        ComputeEvent containts Sub_line_l(p), Sub_line_u(p), Sub_line_i(p)
        STAT=Select S ε PT
        IF Sub_line_l(p) ∪ Sub_line_u(p) ∪ Sub_line_i(p)
            Intersection_Point = PT
        END IF
        IF Sub_line_l(p) ∪ Sub_line_i(p)
            Delete(Intersection_Point)
        END IF
        IF Sub_line_u(p) ∪ Sub_line_i(p)
```

```
                Compute $S_{left}$ and $S_{right}$
            END IF
            Intersection_Points = Intersection_Point  //storing all intersection points
        END IF
END FOR
```

In case of handling of each case after Sub_line_u(p) ∪ Sub_line_i(p) is found not to be empty then each and every intersection point with respect to the left and right neighbors need to be computed and thus the order needs to be changed and modified until there no longer exists any more event points.

The running time of the algorithm has found to be O(n+I)$\log_n$ where I represents the number of Intersection points. The queue is assumed to be in in the form of a binary tree and that is found to take O(n$\log_n$) time. Handling all the events that include insertions, deletions and finding adjacent lines takes O(logn) time each. If the total number of line segments when calculated linearly is m(p), then the running time is found to be O(mlogn). As and when the line segments are detected and removed, using graph theory with Euler's formula it is found to be O(n+I), as the intersction point of two segments must be deleted when they stop being adjacent in which case the total time is calculated to be O(nlogn)+O(Ilogn).

5.2 Half Plane Intersection

As elaborated previously, the aim is to find a point lying on the feasible region in conjunction with all the lines. Now, instead of considering each boundary as lines, it is important to consider them as planes with a directional vector. In accordance with the directional vector the feasible region with respect to a plane would therefore be the right side of the respective planes. In order to find a point that would satisfy n constraints at the same time, it is expected to be in the bounding line of some plane. The common feasible region would contain the point on any of the bounding lines. The following figure (Figure 5-4) depicts the different types of intersection planes that are being explained here. The polygonal region obtained by intersection of the edges is found to be convex with respect to the constraints used.

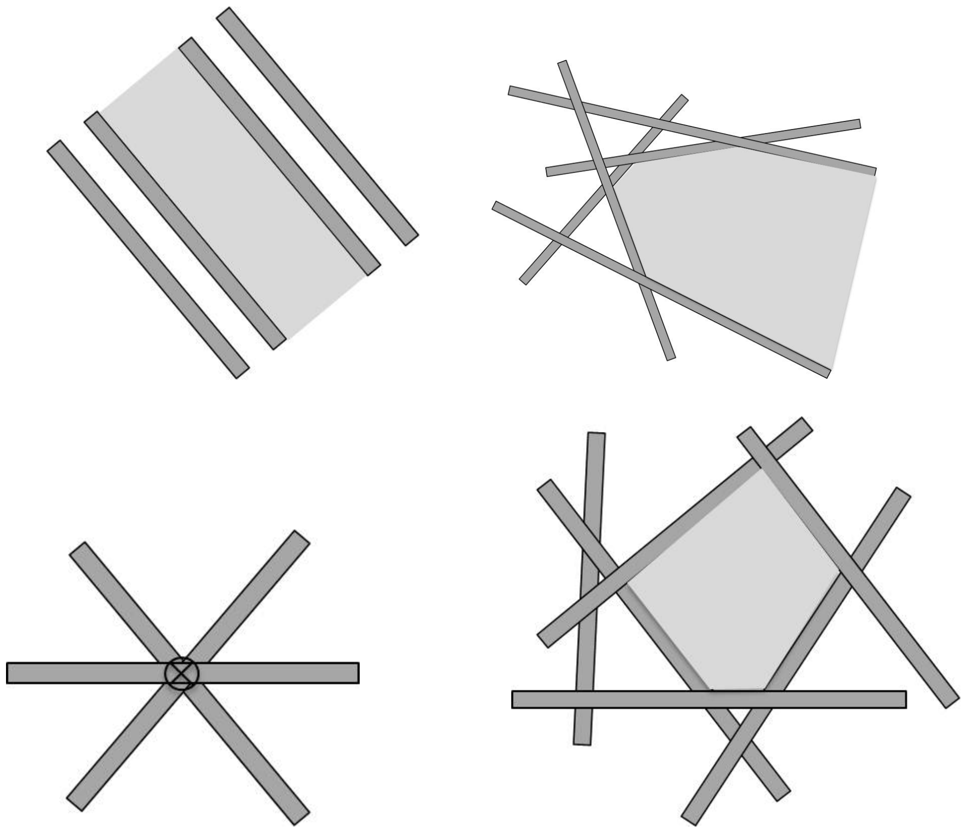


Figure 5-4 Examples [38] showing the feasible region among half plane intersection

We proceed to solve the problem of intersection of n half planes and then finding the point on minimum. After studying the problem in depth and by looking to the examples above, it can be concluded that in order to apply an algorithm, the region obtained can be open or that all the planes can converge to a single region. However as we saw that the algorithm elaborated above is Plane Sweep Algorithm that allows us to find all the intersecting points when a number of planes intersect each other. In this case as we only require the one point on the minimum side of the common feasible region, the above algorithm needs to be modified in such a case.

An approximate divide-conquer concept needs to be applied in case of n constraints and also to take care of the special case as elucidated above. The algorithm pseudo code is given below:

```
INPUT: H //set of n half planes
IF  card (H) =1
        C= SelectUnique(H) //function to select unique half plane from H
ELSE
        (H1, H2) = Split(H) //into H1 and H2 of size ┌ n/2 ┐ and └ n/2 ┘
        C1=H1
        C2=H2
        C=Intersect(C1,C2) //intersecting regions of C1 and C2
END IF
```

The INTERSECT CONVEX REGIONS is the algorithm that is explained above in the pseudo code. The Plane Sweep Algorithm finds all the intersection points within all the planes using a binary tree search. So, applying the same analogy the aim is to find intersection point within two edges in C1 and C2 by recursion and then procure the point at minimum that satisfies all constraints. So finally the computation of intersection of C1 and C2 takes O(nlogn) time where its assumed that it would be an intersection of n half planes and therefore has at most n edges and vertices. Since now we know as to what time the computation would take, the next step is to obtain the recurrence for the total running time. It has been found to be T(n)=O(nlog2n). Another important hypothesis here is that the polygonal regions are assumed to be convex. The next step in consideration is the representation of the convex polygonal region in order to compute the intersection region. In order to do the same, an organized list is maintained for the left and the right side of the intersecting plane. This explains as to why the regions C1 and C2 are considered to be convex, as this would allow us to store the edges in simple data structures rather than in complex ones. As there as to be 4 sides, its further elaborated as left_edge_C1, right_edge_C1, left_edge_C2 and right_edge_C2. The basic approach in such a scenario would be to make use of a pointer. However, since the platform here is MATLAB there is an initial set boundary of 4 edges and with each intersecting orca line. Each edge of the boundary is stored as a matrix. As each ORCA line intersects the boundary, the subsequent feasible edges are found and thereby the region is modified. As we have seen before in the identification of feasible regions, there can be cases where there the ORCA line and the set boundary do not have feasible regions. Such a case can be referred to as the global deadlock where the robot is expected to stay with velocity zero as the movement in any direction can cause collision. Hence the plane sweep algorithm has been modified with limitations in order to satisfy the requirement for finding a consistent optimal velocity for multi robots.

Chapter 6

Selection of Optimization Velocity

For setting optimization velocities of the robots, there are various options with respect to each scenario. There can be situations where $V^{opt}{}_{A}$ can be set to zero or $V^{opt}{}_{A}$ is $V^{pref}{}_{A}$. In this section, we elaborate on the various choices that would be available according to the orientation of ORCA half plane and also on the basis of the current positions of A and B. The first scenario is where $V^{opt}{}_{A}$ would be set to zero when ORCA is not empty. It would set a velocity for all robots around A that would make sure that all of it is on a collision free path for time τ. On observing the Figure 6-1 we see that robot A is surrounded by B, C and D. In this situation it might be more apt for A to remain in its position and wait for others to move away. However, for all other robots the point 0 always lies outside the $VO_{A|B}$. Another important observation to be noted here is that the line delimiting $ORCA_{A|B}$ is perpendicular to the line connecting the current positions of A and B. One of the disadvantages of selecting velocity as zero in such cases is that in cases of dense scenarios then it could result in global deadlock. Another instance is where is the optimized velocity is set as $VEL^{pref}{}_{A}$. Ideally this should work in case the preferred velocity is set to an optimal velocity. On the contrary in high-density conditions, the preferred velocity would be set to a high magnitude, which would make the linear program infeasible. As a result of which there would unreliable navigation of the robots in the current environment. There is also a probability of the same situation happening in medium populated environment.

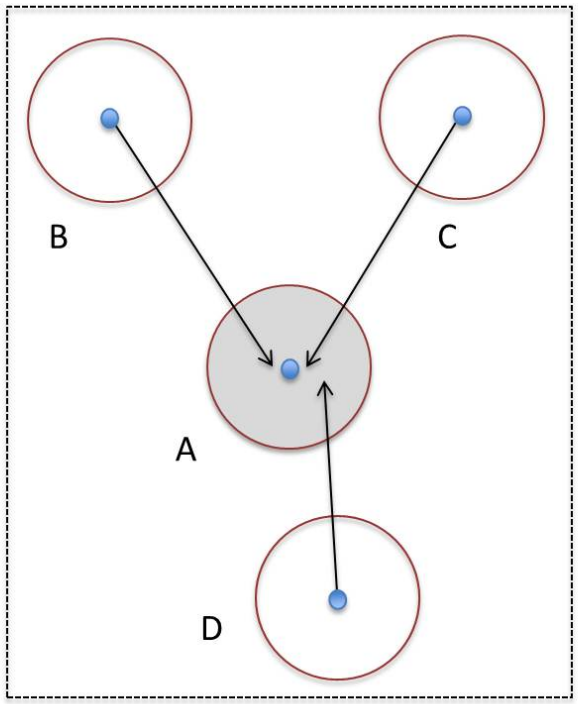


Figure 6-1 Robot A is placed in between three other robots. All of the robots are found to be heading towards A. In such an instance an optimal velocity should be found for A that would allow it to move without colliding with any of the surrounding robots.

Another option is to set the optimal velocity as the current velocity, which would characteristically point to the facts discussed earlier that all the robots would automatically adjust to the situation. This is mainly because all of them would be moving in the desired velocity through a collision free trajectory. Another issue rampantly observed is that there would be still infeasible regions (refer Figure 6-1 and Figure 6-2) selected and thereby results in the velocity being set to zero in crowded scenarios. This is where the term “safest possible velocity” comes into prominence. As it can clearly seen in Figure 6-1 choosing VEL^{opt} might not always be the best option. So instead of choosing a velocity that conforms to all the conditions with respect to the surrounding environment, the better option is one that minimally commits to the constraints instilled by the rest of robots. Such a velocity is known as safest possible velocity which is selected based on the minimum distance to the boundary of the plane ORCA $^{\tau}{}_{A|B}$. Suppose the current velocity lies within ORCA $^{\tau}{}_{A|B}$, the Euclidean distance is found to be negative. The mathematical representation is

$$VEL^{new}{}_{A} = \arg\min vel \in D(0, VEL \max A) \max B \neq A \text{ dist } A|B(VEL) \qquad (15)$$

The inspiration for the minimal distance comes as a result of the objective that this could easily be extended to three-dimensional linear program. Similar to finding a minimum distance here, the point (VEL*,dist) with minimum distance above the plane is selected. The running time is expected to be O(n). An important observation to be made here is that the selection of velocities for robots is more or less dependent on the surrounding robots and thereby enables the entire environment to remain consistent.

In the case of static obstacles there cannot be division of responsibility to avoid collisions. To move through collision free trajectory around a static obstacle within τ time,

$$VelOb^{\tau}{}_{A|O} = \{VEL \mid \exists t \in [0,\tau] :: t \quad VEL \in O \oplus - D(POS_A, rad_A)\} \qquad (16)$$

Here O is the set of line segments modeled based on the static obstacle and the velocity needs to selected that would be outside VelOb $^{\tau}{}_{A|O}$ and thereby selecting a point inside the complement of VelOb $^{\tau}{}_{A|O}$ should help us attain our objective. On the contrary this would not be a convex region on which applying the linear program would be difficult. Hence it would be appropriate to set the velocity to zero. In crowded conditions increasing the value of τ(refer Figure 6-3) would help as it almost makes the robots to move towards the obstacle so as to not come in the way of other robots. However this is in lieu that the robots remain within the tough constraints as computed through the algorithm. Instead of the robots moving in all directions around the obstacle they do move to their respective target position as their trajectory is continuously computed.

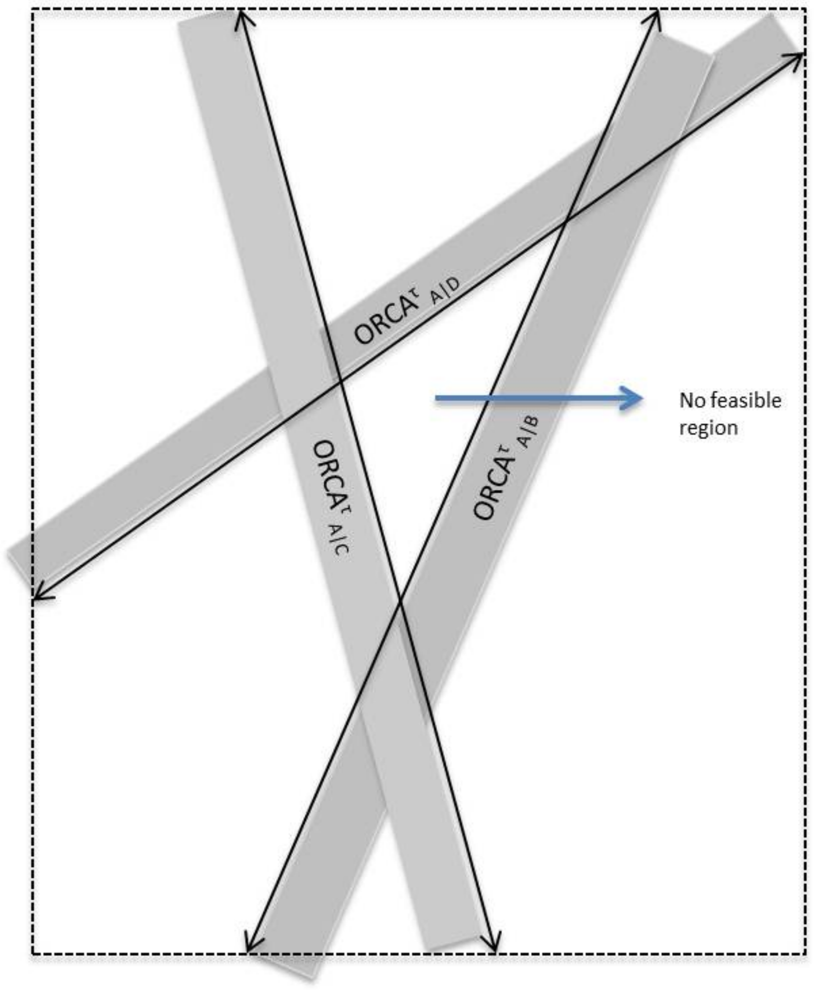


Figure 6-2 If no optimal velocity can be computed for robot A then empty set is obtained. In such a case there would be collision within the defined time frame.

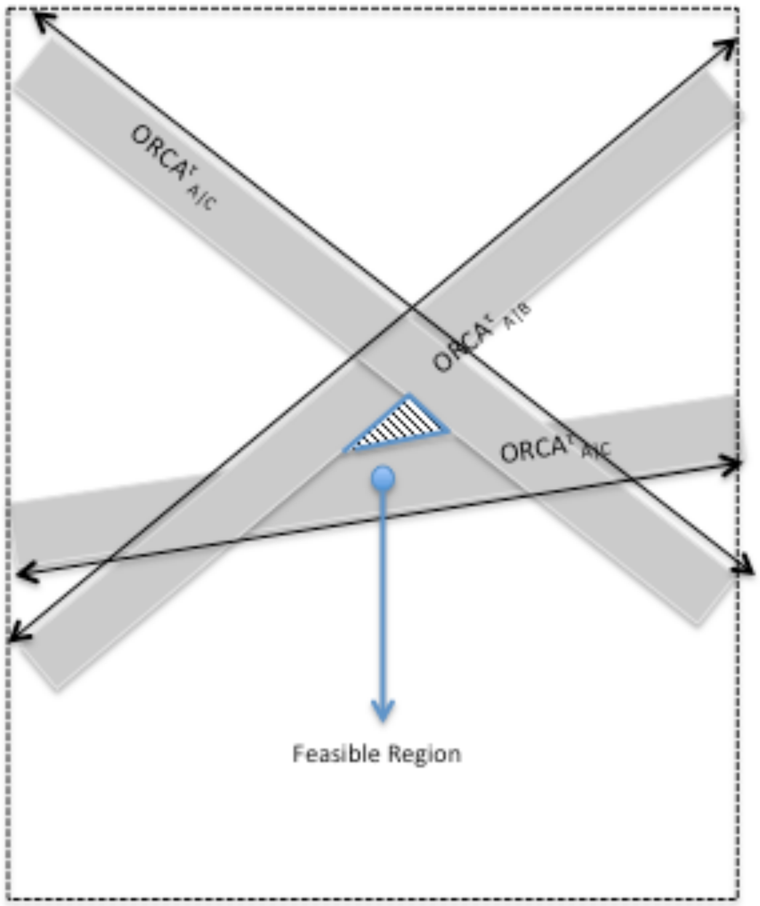


Figure 6-3 Feasible region is obtained once the time window is increased and thereby an optimal velocity can be obtained from the feasible region.

Chapter 7

Collision Avoidance with Particle Filtering

7.1 Problem Definition

The main objective of the problem is to redefine the assumptions as already proposed by the existing ORCA algorithm. Perfect sensing was one of the main assumptions put forward by the ORCA algorithm. In real-life scenarios, this is not always the case as there is always a high probability of obtaining inaccurate measurements from the sensors. In order to address such issue and thereby make the algorithm more effective for real platforms, a filtering approach was introduced with uncertainty in measurements of velocity.

7.2 Applying Particle Filter to collision avoidance

A majority of the applications in robotics require estimation of the system accurately over a period of time. Most of the current research problems would require the dynamic system model to process the data as and when the sensors acquire measurements [38]. Obtaining the measurements and at the same time computation for the specific model is significant as the system has to be calibrated with respect to the new data that the sensors are collecting over the period of time.

Particle Filters are sequential Monte Carlo methods that comprise of approximate techniques to calculate posterior densities, which are basically point-based distribution. As far as our system is concerned, it is mainly a dynamic model that keeps rapidly changing over the period of time. The progression of the system would be best described with a discrete time approach since the measurements of the data are collected at distinct cycles. Furthermore the system would be modeled on a state space approach, as it would also need to keep track of the noisy measurements that are also obtained with respect to a time series model. Also the prime advantage of the state space approach is that it allows flexibility in handling linear as well as nonlinear systems.

One of the things to consider while conceiving a dynamic system model is that it needs to be broken down into two main models. Initially a model needs to be generated that would illustrate the development of the state with respect to time: System Model. Also another model that is to be considered based on the noisy measurement of the state: Measurement Model. In order to determine the Probability Density Function (PDF), the formula given helps in giving an insight for an unknown time varying function.

$$p(x^t|Y^t) =p(y^t|x^t)p(x^t|Y^{t-1})/p(y^t|Y^{t-1}) \qquad (17)$$

Here PDF of the state at time t based on previous measurements $Y^t = \{y1,y2,y3\ldots\ldots y^t\}$.

A main assumption while applying a Bayesian approach is that the probabilistic forms of the model also need to be contemplated. During the process of elaborating the PDF of the dynamic model in consideration, all the available data are encapsulated so that an optimal estimate of the state may be obtained. Also one another important measure that needs to be calculated is that accuracy of the state. During most of the instances, when such an approximation needs to be made a recursive filter is used in which rather than accumulating all the data and then computing, here the process is carried out as and when the data arrive in each cycle. Recursive Filter fundamentally contains two main phases: Prediction and Update. The Prediction stage with respect to the dynamic model that is being discussed here would essentially predict the next state PDF for the consecutive cycle of measurements. As the random noise is also been taken into account, the prediction phase would also transform the PDF with respect to the noise in the measurement. The prediction stage is followed by the update stage where the anticipated PDF is to be altered according to the latest measurement obtained. In order to ascertain the values for the expected PDF there is a high amount of computation power required for parallel applications, ([39]) in order to implement Monte Carlo Methods. On the basis of weighted sum computed using the delta functions, particle filters approximate the PDF. Considering the target space, {wt (r) t-1,x (r) t-1} for r = 1….N that would showcase the samples based on the targeted PDF.

$$p(x^{t-1} | Y^{t-1}) \approx \sum \text{Ns r=1 w r k-1 } \delta(\text{x t-1} - \text{x r t-1}) \qquad (18)$$

Here w r t-1 is the weight with respect to the r^th observation point and also ∑ Ns r=1 w r k-1 =1. As discussed above the particle filtering approach is spread over two steps. Initially, all the points obtained by the updated system at time t-1 are transformed to the cluster of points that would describe the predicted density, $p(x_{t-1} | Y_{t-1})$. So now the next step would be to update the group of particles obtained through the propagated density, $\{w^{(i)}{}_{t-1}, x^{(i)}{}_{t}\} \sim p(x_t|Y_t)$ to the set of particles that contain the updated weights. Such a procedure is known as sampling and here the aim is to approximate the incognito updated density. In order to simulate from such a distribution, we sample the points from the available distribution and assign the weights to them so that sample can be simulated from the unknown distribution.

Hence the required weights for the updated points are :

$$w^{(r)}{}_{t} = (w^{(r)}{}_{t-1}\, p(y_t|x_t{}^{(r)}) / \sum^{N}{}_{l} = 1\, w^{(l)}{}_{t-1}\, p(y_t|x^{(l)}{}_{t}) \qquad (19)$$

where $p(y_t|x_t{}^{(r)})$ is the measurement likelihood function that would showcase the target state with the rth particle $x^{(r)}{}_{t}$. Another issue found in such systems is degeneracy. One needs to ensure that the equal weights are assigned and not one particle stands out with a high weight number assigned to the same. In such situations the only solution is to resample the points and thus obtain particles with equal weights.

Although there are various filtering approaches that would cater to the requirement, a particle filter was considered the best one to use. Since the system that is in consideration is a dynamic system, a particle filter was considered, as it would allow one to use a complex model and thus obtain approximate measurements instead of accurate ones. Using a Particle filter, a collective set of best estimates of velocity of the current body is obtained. The set of velocities can be regarded as virtual set of bodies that are moving in different directions with respect to another body.

Chapter 8

Experiments and Results

The simulation environment is defined in the form of a square within which the robots are expected to move around. To implement Reciprocal Collision Avoidance Algorithm, robots are designed in the form of circles in the respective simulation environment. The simulation environment is implemented in MATLAB on Windows 7 with 64 bit OS and 4GB RAM. In order to implement all the dimensions of the algorithm, the process has to be broken down into several steps. Initially the simulation environment had to be set up within which the robots should be able to move smoothly. Another module had to be set up within which the velocity obstacle is worked out, which would give us the set of collision avoiding velocities. After obtaining the set of collision avoiding velocities, the next step is to select the optimal velocity. In order to do the same, linear programming model is developed, that adds ORCA planes continuously and simultaneously calculates the feasible region. Once all the modules are set up, test cases are developed that would showcase different scenarios. Each of the sections is elaborated below.

8.1 Setup the simulation environment

The first step is to set up a defined boundary and have the robots navigate randomly within the set boundary. The robots are constructed with a definite radius and initialized with current positions as shown in Figure 8-1. The positions of each robot are continuously updated but still ensured that they stay within the boundary. Here the expected movement is in the form of bouncing motion among the robots.

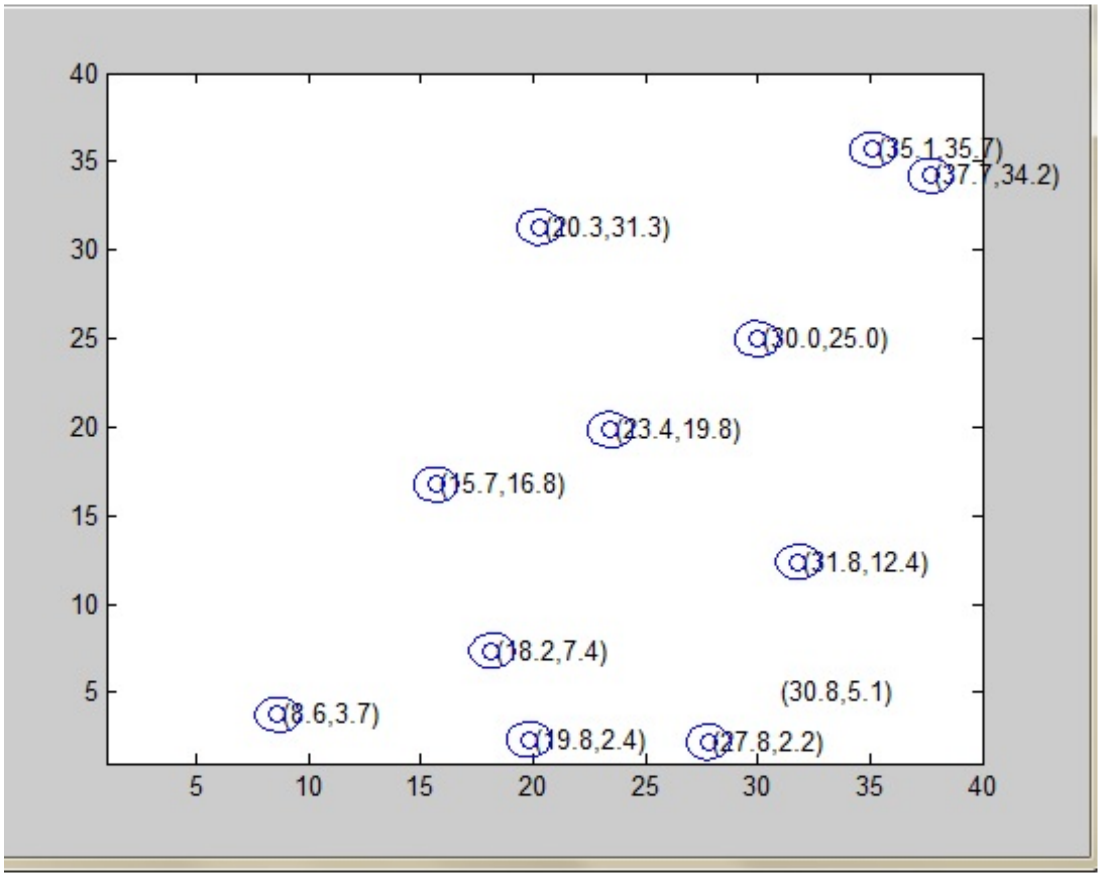


Figure 8-1 Simulation set up with positions updated continuously (horizontal axis: Vx, vertical axis: Vy)

8.2 Obtain the Velocity Obstacle of one robot with respect to each other

After the movement of the robots is worked out, the next step is to compute and calculate the velocity obstacle between two robots. In order to geometrically depict the velocity obstacle between two robots, we first need to obtain test cases where the robots are expected to collide within the next time frame. To achieve this, the robots have to be moved towards a target position and test cases have to be obtained such that one would come in the path of each other as shown in Figure 8-2.

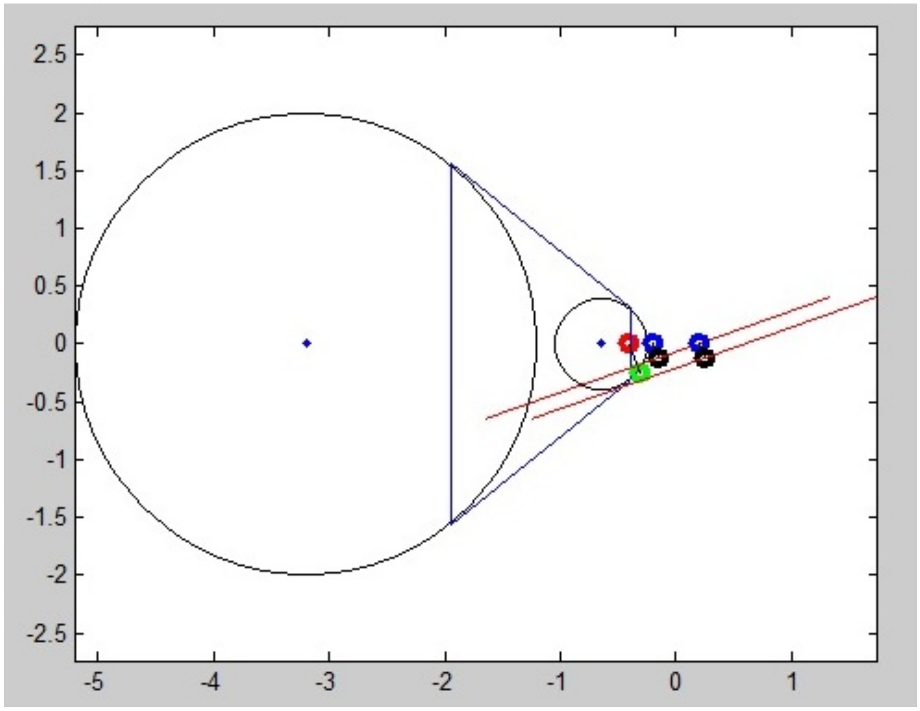


Figure 8-2 The velocity obstacle is obtained when a robot A is expected to collide with robot B. In the above snapshot ORCA lines are being constructed as and when the robots are close to each other. The velocity obstacle is constructed with the specific parameters according to RCAA. The closest point to the boundary is obtained with which the ORCA lines are constructed.

8.3 Implementing the optimization algorithm for velocity selection

Once the velocity obstacle is obtained, selection of appropriate velocity has to be made from the ORCA lines. Choosing the velocity has to be made appropriately such that robot is made to deviate only so that it avoids collision but it reaches the target position. A desirable solution is found using a linear programming approach and is shown in Figure 8-3 and Figure 8-4. The ORCA planes contain alternate velocities that would avoid collision between the two bodies. To select a velocity that is close to the preferred velocity, the minimal point is found. The right side of each ORCA plane is treated as a feasible region. The computed ORCA plane is made to intersect the boundary. Hence, minimal point is found by taking the shortest distance to the preferred velocity.

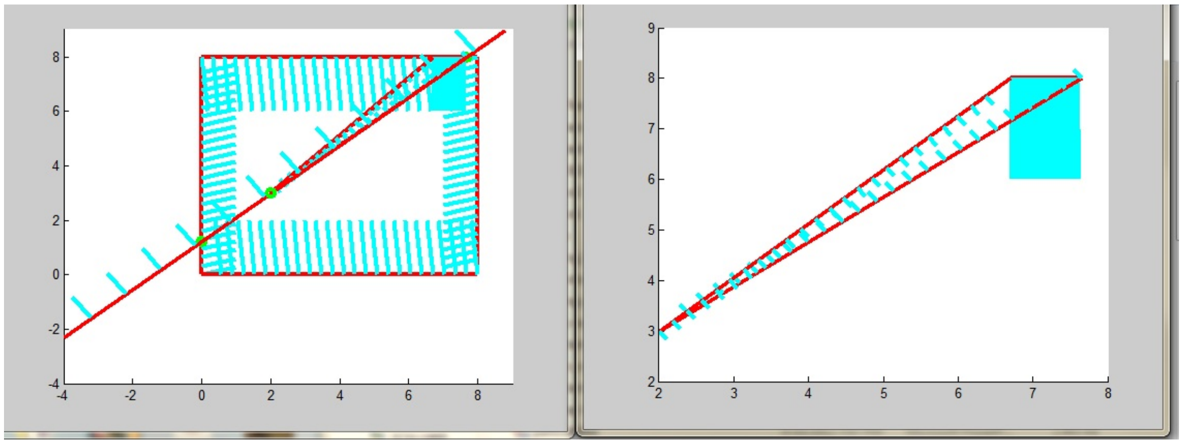


Figure 8-3 Snapshot of the Test case where A is expected to be collided by B and C in time t. At such an instance the set boundary is intersected by two ORCA planes of A with respect to B and C. Applying linear programming, the feasible region is computed (horizontal axis: Vx, vertical axis: Vy)

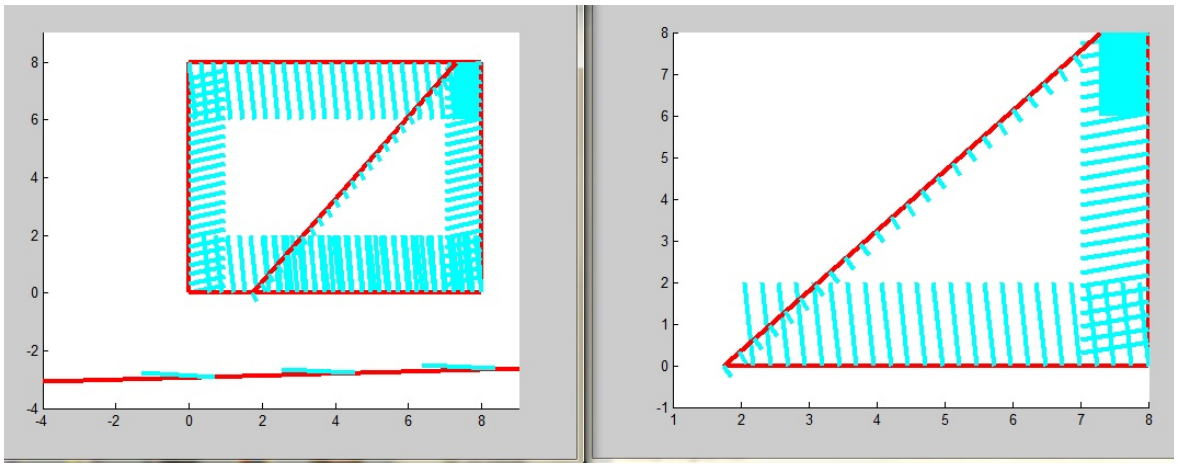


Figure 8-4 Snapshot of the Test case where one of the ORCA plane lies completely outside the set boundary. In such an instance, the favorable plane selected is as to whichever lies on the right side (horizontal axis: Vx, vertical axis: Vy)

Here the aim is to iteratively add the ORCA lines generated by the robots with respect to each other and thereby find the feasible region from which the minimum is found. As the robots keep moving it is important to simultaneously obtain the velocity obstacle and thereby calculate the optimal velocity within the time frame.

8.4 ORCA applied to one robot among n-robots

Initially ORCA is only applied to one robot and not to other robots. A trail path was constructed, so as to trace its trajectory. The test case was used where the two robots, shown below are going to collide in time t. A case was developed (shown in Figure 8-5) where they would swap positions with each other. One of the robots is expected to sense the incoming obstacle and deviate enough to avoid and then get back to its original path.

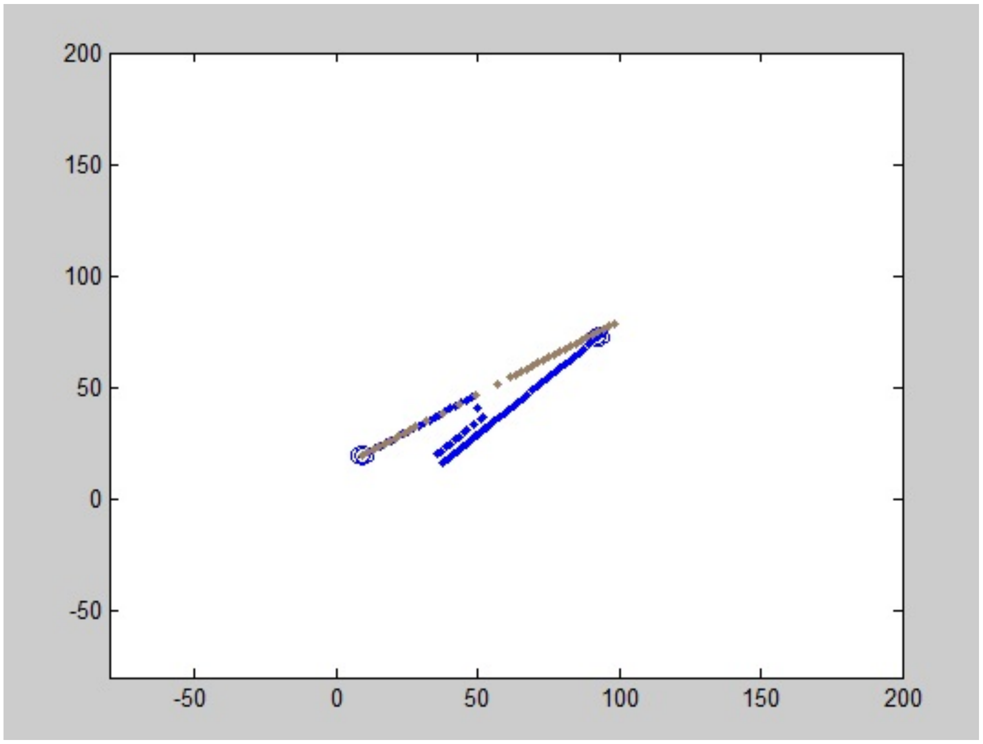


Figure 8-5 Snapshot of the Test case where the robots are moving towards its target position and expected to collide in time t. ORCA only applied to robot A and not B. Here only one of them changes the path when they are switching positions with each other (horizontal axis: Vx, vertical axis: Vy)

After testing with two robots, slowly the number of surrounding robots was increased so to observe the behavior with different time windows and increasing number of robots. Here all the robots are assigned initial and target positions, however the ORCA algorithm is only applied to one of them. The aim of the robot A was to sense all the 10 robots and deviate to avoid collision but eventually reach its target position. The test case was set up such that all the robots were moving towards A. This is shown in Figure 8-6.

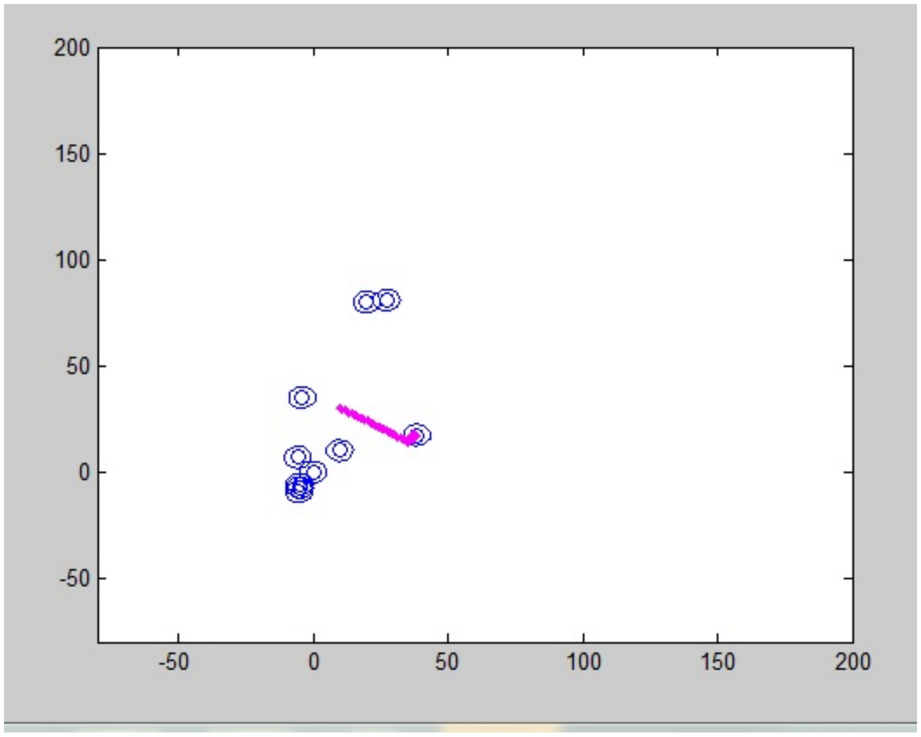


Figure 8-6 Snapshot of the ORCA only applied to robot A and not to other 10 robots. Only the respective robot change path without colliding with the rest of the robots (horizontal axis: Vx, vertical axis: Vy)

8.5 ORCA applied to n robots

The next functionality of ORCA to be tested was applying the algorithm on all the robots moving in the simulation environment. The responsibility was expected to be shared between the two robots. A test case was set up similarly where two robots are swapping positions with each other and a collision was expected in the time t. As shown in the snapshots below both are moving towards their target position but are expected in collide in the next time frame. Due to the application of the collision avoidance algorithm, both the robots deviate from the path to avoid collision but finally reach the target point. This case is shown in Figure 8-7.

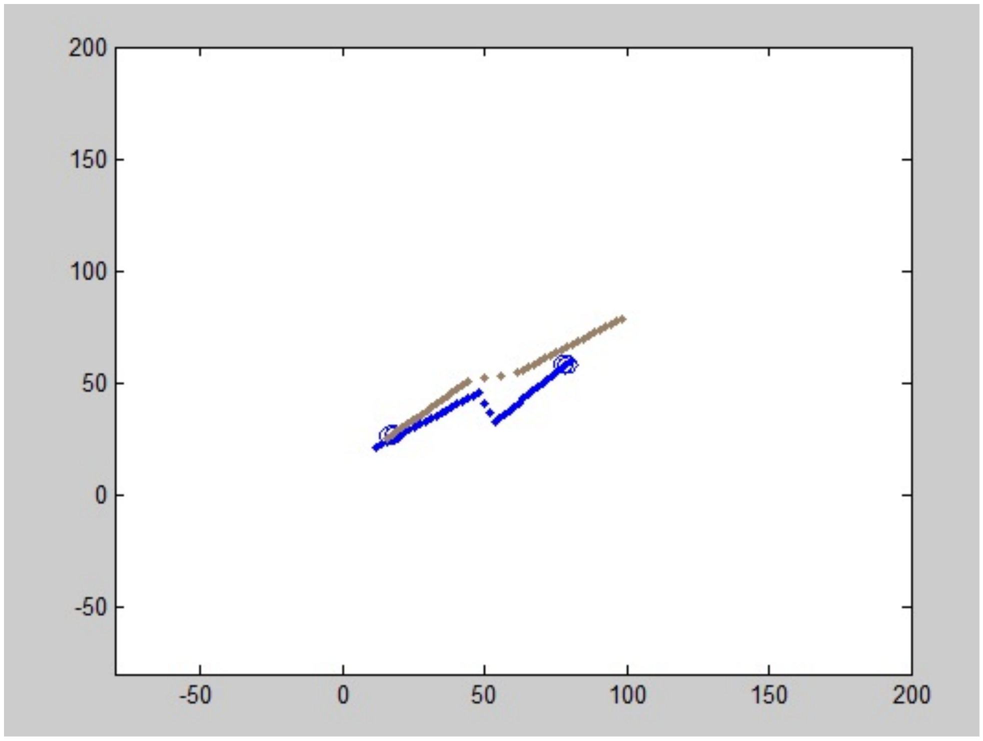


Figure 8-7 ORCA applied to robot A and B. The robots are moving towards each other but move away from their original path to avoid hitting each other. After deviating from its original path, it moves back into its initially calculated trajectory to reach its target point (horizontal axis: Vx, vertical axis: Vy)

8.6 ORCA applied to robots with Particle Filter

To test our proposed approach Particle filter was applied to velocity estimate. Applying particle filter to the velocity provided a wider range of velocities to be selected from for collision avoidance. After calculating the preferred velocity, using a generic particle filter, velocity was estimated. Further after calculating velocity obstacle and using linear programming optimal velocity was obtained. The snapshots of the test cases, after applying particle filter to the existing collision avoidance approach are shown below in Figure 8-8, Figure 8-9 and Figure 8-10.

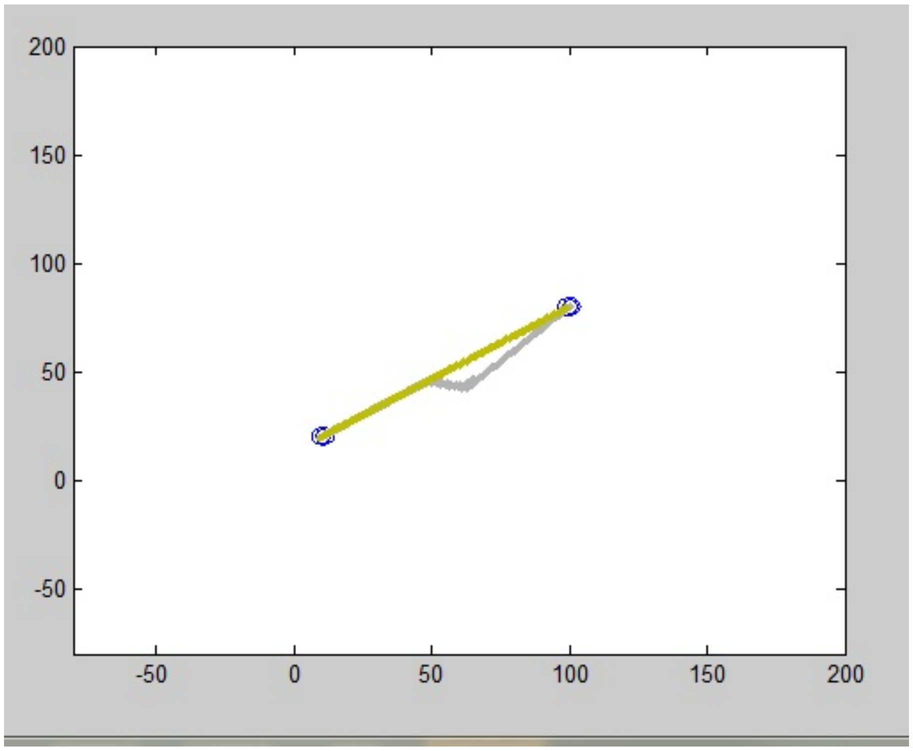


Figure 8-8 ORCA applied to robot A and not B with particle filter sampling. Here the velocity is estimated using particle filter approach. Once there are more velocities to select from, a more controlled and streamlined motion is observed. Here robot A and B are switching positions with each other (horizontal axis: Vx, vertical axis: Vy)

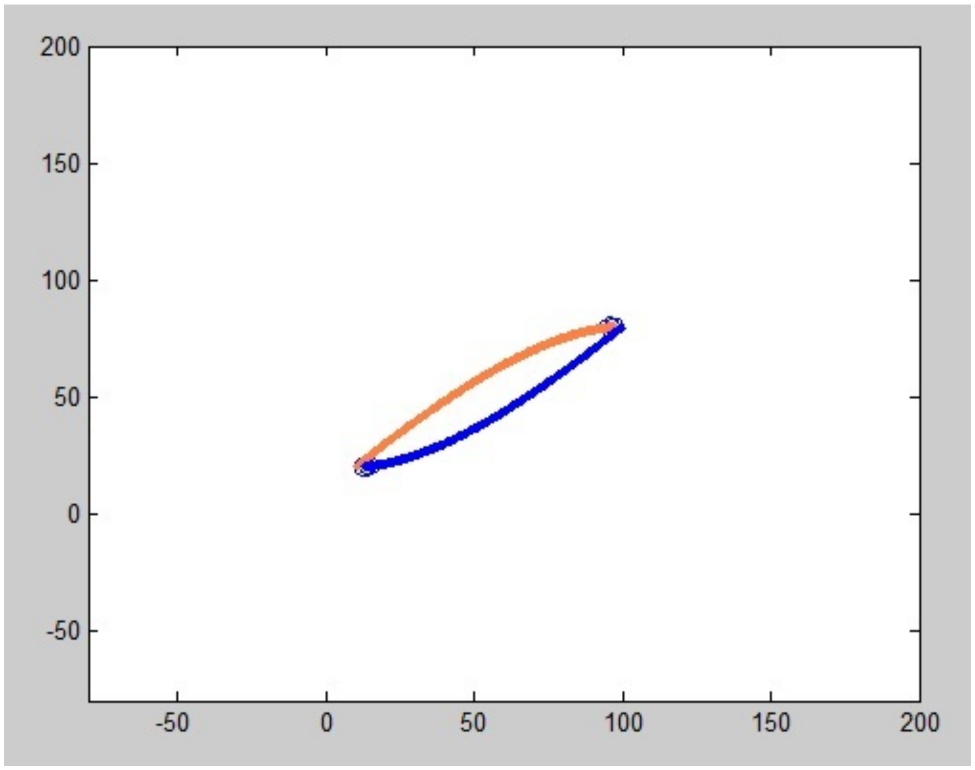


Figure 8-9 ORCA applied to robot A and B with particle filter sampling. Here robot A and B are switching positions with each other (horizontal axis: Vx, vertical axis: Vy)

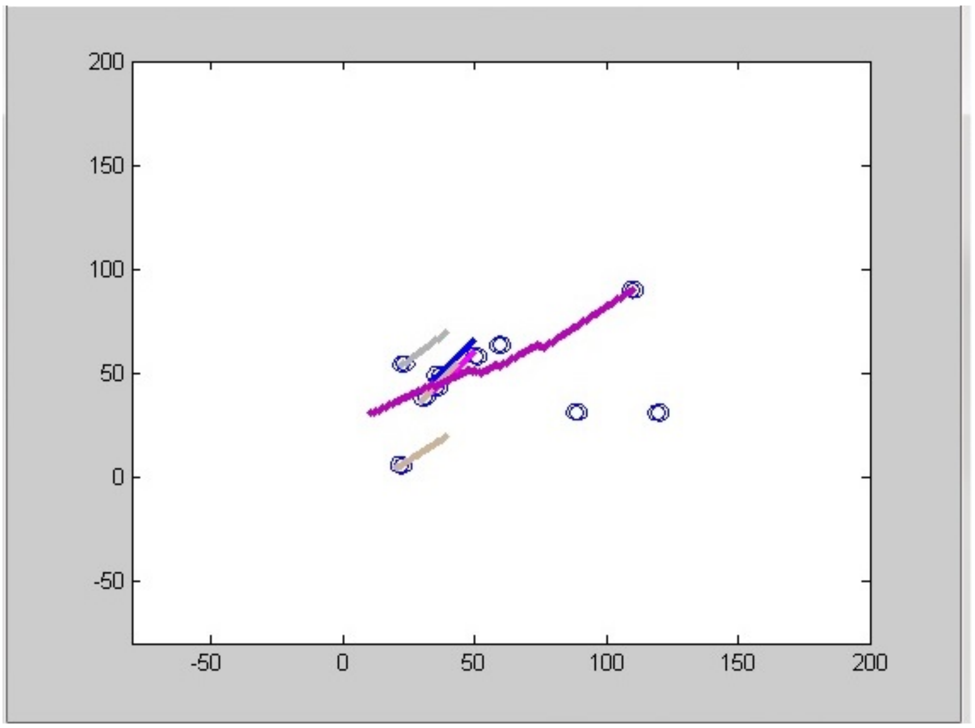
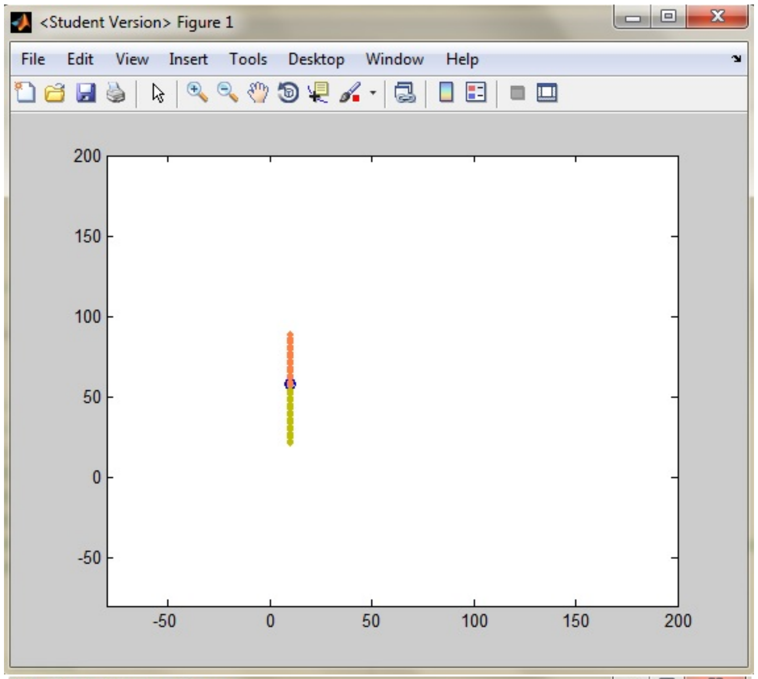


Figure 8-10 ORCA applied to robot A only with particle filter sampling. Here the velocity of A is estimated based on the defined sampling rate. The other robots are moving towards its target position but are placed such that all of them are moving towards A. The objective of A is to avoid each of it and move towards its target position (horizontal axis: Vx, vertical axis: Vy)

As we see from the above figures after applying particle filter there is more streamlined motion and better avoidance. One limitation while applying ORCA was that the algorithm was found to fail in with a very small time frame. In order to counter such situations particle filter was applied and concrete results were obtained were it was found to work in such situations. In the following snapshots (Figure 8-11 and Figure 8-12), the results have been elaborated.



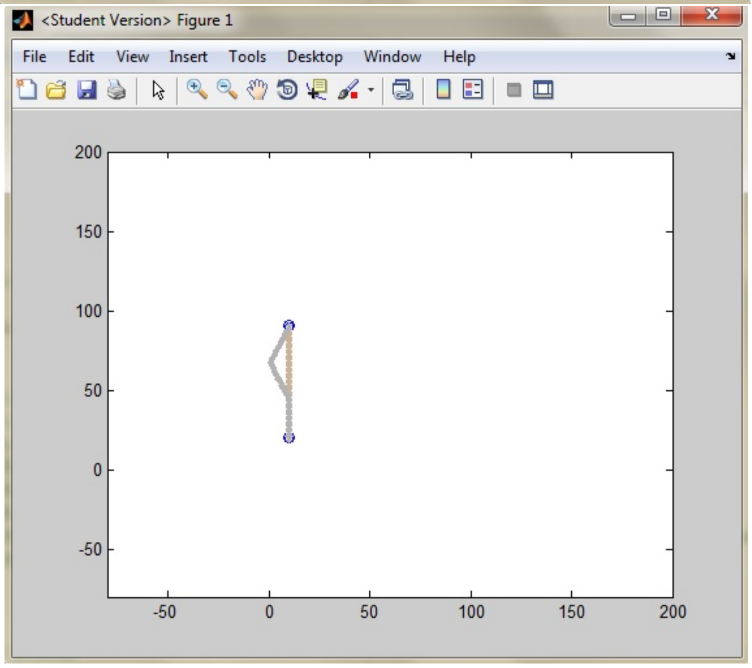


Figure 8-11 ORCA applied to robot A only with particle filter sampling. A very small time window is applied within which it was found to collide when only ORCA was applied. However after particle filter was applied it was seen that one robot was able to avoid colliding with the other robot. (horizontal axis: Vx, vertical axis: Vy)

Similarly ORCA was applied on all the robots to cases where it was found to fail with a small time window. In such cases applying particle filter enabled robots to avoid each other. This is because in the existing approach without particle filter, small time windows

(0.05 to 0.35) were too less for the robots to use the velocity to process the collision avoiding velocity profile. Using particle filter in such instances gave flexibility to select alternate velocities and perform collision avoidance. In those cases where the ORCA approach was only applied to a single robot in the system, the robot was able to detect the incoming obstacle and steer away itself to avoid collision. In cases where the approach was applied to all the robots with particle filter, one of the robots was able to estimate the velocity earlier and deviates itself enough to avoid the incoming robot. Since each of the robots is an intelligent decision-making entity, as one of the robots moved away before, the other robot doesn't detect any robot in its trajectory. Therefore the robot does not sense the need to take a detour unnecessarily. Hence the robot moves along its expected trajectory and eventually reaches its destination point. Another observation during testing was that while applying the existing approach and the trajectory was being traced out without particle filter, there were break points seen during collision avoidance. However while applying particle filter, the trail was seen without any breakpoint indicating there was streamlined motion during navigation.

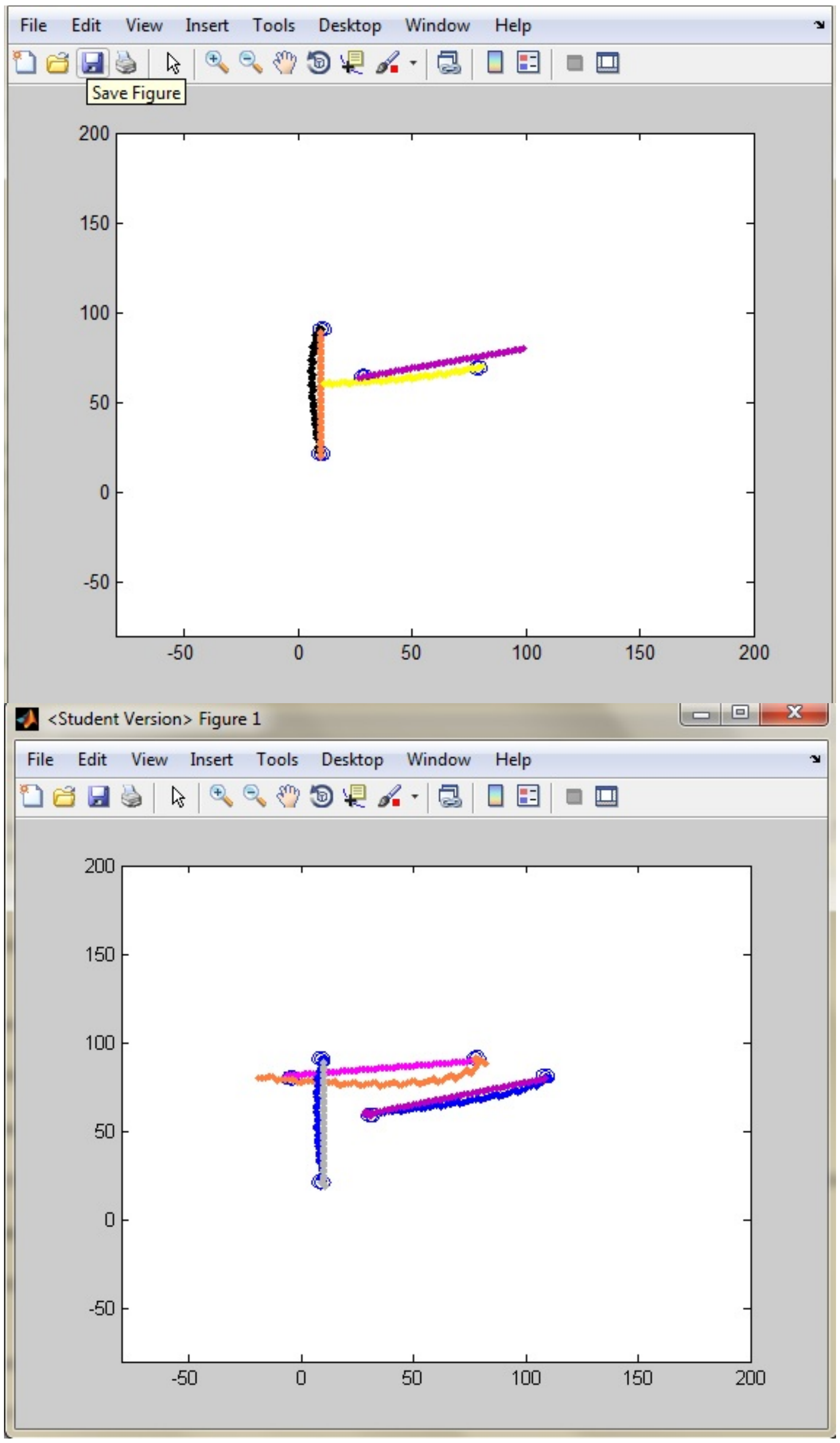


Figure 8-12: ORCA applied to all the robots only with particle filter sampling. Due to particle filter sampling, one of the robots would be able to sense and estimate a better velocity while the other robot was only required to deviate minutely from its path A very small time window is applied within which it was found to collide when only ORCA was applied. Better results were obtained with more than two robots as well (horizontal axis: Vx, vertical axis: Vy)

In order to further elaborate the results, statistics was obtained in the form of collision percentages. This was calculated based on the proximity between the two robots. First the distance between the centers of the robots was obtained. This is then compared to see if the distance was less than the combined radii between them. If it is then the robots have overlapped with each other and this is considered as collision in our experiments. Testing was done on a system with more than two robots and collisions were compared with and without appending particle filter. The percentage of robots colliding with each other in these approaches was compared. The collision percentage was obtained for a wide range of test cases with each time window. Anywhere between four to six robots were used in each of these test cases. In addition, the test cases were required to apply ORCA alone and with particle filter on one robot and multiple robots. Then the average collision percentage was calculated for each case with and without particle filter. This is presented in Figure 8-13 and Figure 8-14, which clearly prove that the collision percentage is significantly reduced using our method than with going with just ORCA. Not only this, ORCA is only applied on one robot in a system of robots.

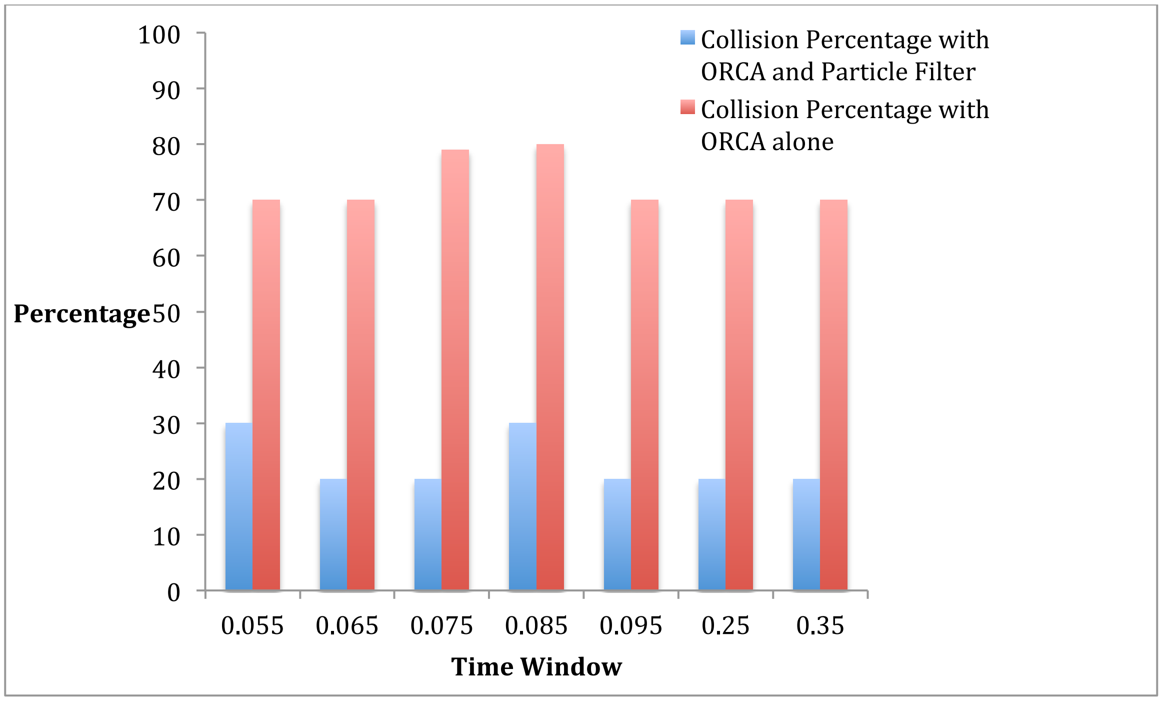


Figure 8-13: In this graph x axis represents the time window that is used and the y axis shows the collision percentage achieved. Here the green color represents the collision percentage obtained after applying particle filter with the collision avoidance approach. However the red bars represent the high percentage of collision with just the collision avoidance approach. This data is with respect to various test cases when ORCA is applied to one robot against all other robots. A more consistent result was found in this case as opposed to ORCA applied to more robots

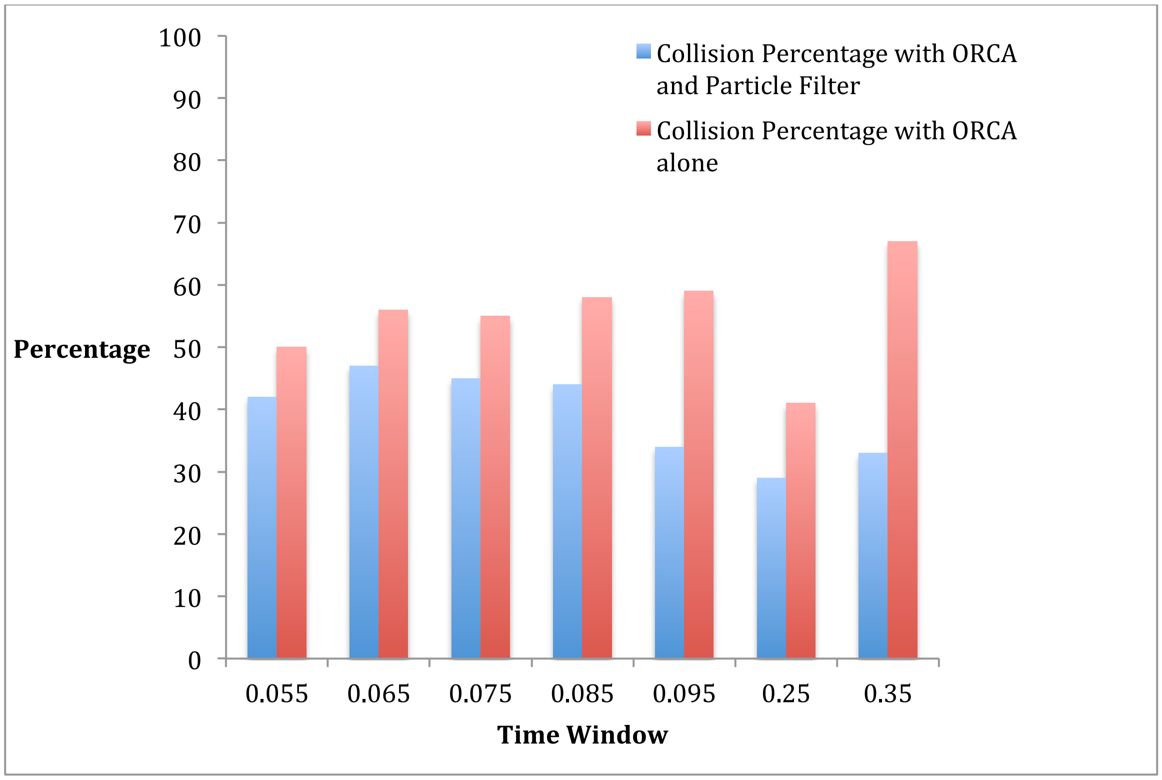


Figure 8-14 In this graph x axis represents the time window that is used and the y axis shows the collision percentage achieved. Here the green color represents the collision percentage obtained after applying particle filter with the collision avoidance approach. However the red bars represent the high percentage of collision with just the collision avoidance approach. This data is with respect to various test cases when ORCA is applied to all robots.

Also the sampling rate done between 10 and 30 resulted in no significant difference on the results. Stress testing was done on specific test cases with very small time window where the collision approach was not found to work. In such instances using particle filter made the system more stable and there was a lesser probability for collision to occur. Also in some instances imaginary values or empty set were assigned for velocities, with just collision avoidance approach. Applying particle filter gave a broader spectrum to select velocities.

Chapter 9

Summary

9.1 Conclusions

One of the main assumptions for ORCA [1] algorithm is that there is always perfect sensing for all robots when the algorithm is applied on the robots all the time. However in order to implement this approach more realistically, we have introduced particle filters. This would allow resampling of velocities and thus provide more scope for selection. One of the main improvements seen is that there was less oscillation observed in robot navigation thus providing more stability for the system. Also the number of collisions seen among large number of robots was less with particle filter. Furthermore, using a rectangular boundary for linear optimization provided more scope in various time windows. Applying a modified approach provided a more consistent breadth for the ORCA planes with respect to alternative velocities for collision avoidance. Another capability added here that was not present is adding an error percentage to the measurement which would allow one to translate this concept easily on a real platform. In order to get more room for alternative paths for robots not to come in the way of each other in a crowded scenario, efficient and consistent solutions are developed to ensure continuous smooth movement of robots.

Through this thesis, we have enhanced our understanding of the domain and have made contributions to this research domain through our simulation program and improving collision avoidance by introducing the particle filtering approach.

9.2 Future Work

Also this technique has been now just tested with a simple generic filter. Moving forward an improvement in this direction would be to apply real time adaptive particle filters. This would provide a more consistent and rigorous methodology for collision avoidance among n-robots. One of the issues while applying the generic particle filter is that it is computationally expensive in the current platform. Also, the application has only been tested with 2D environment and simple kinematic constraints. Furthermore using more efficient particle filters with a larger scope of constraints with respect to 3 D surrounding would provide more avenues in this direction.

There would be considerable challenges while implementing the approach on a real platform. As the mobile robots are assigned tasks that are to be performed while they in motion, it would be interesting to see as to what kind of behavior is exhibited by the robots in such a scenario. Coupling the decentralized collision avoidance approach with another assigned task like collection of data for ecological study is sure to raise a number of real time control issues. These may be related to sensing and processing of that information in real-time and their uncertainties and relaying the feedback such that the robot is able to react in the shortest possible time. The main technical concerns seen during distributed sensing and network control need to be evaluated and appropriate measures have to be taken. Also another issue that needs to be addressed is system stability, which is if one of the robots fail, then the tasks have to be rescheduled efficiently.

References

[1] J. Van Den Berg, S. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," *Robotics Research*, pp. 3–19, 2011.

[2] B. Faverjon and P. Tournassoud, "A local based approach for path planning of manipulators with a high number of degrees of freedom," in *1987 IEEE International Conference on Robotics and Automation. Proceedings*, 1987, vol. 4, pp. 1152–1159.

[3] T. Lozano-Perez, "A simple motion-planning algorithm for general robot manipulators," *IEEE Journal of Robotics and Automation*, vol. 3, no. 3, pp. 224–238, 1987.

[4] O. Khatib and J. F. Le Maitre,"Dynamic Control of Manipulators Operating in a Complex Environment". *In Proc. of the 3rd CISM-IFToMM Symposium on Theory and Practice of Robots and Manipulators*, Udine, Italy, September 1978, pp. 267-282.

[5]D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[6]C. Fulgenzi, A. Spalanzani, and C. Laugier, "Dynamic Obstacle Avoidance in uncertain environment combining PVOs and Occupancy Grid," in *2007 IEEE International Conference on Robotics and Automation*, 2007, pp. 1610–1616.

[7] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 278–288, 1991.

[8] J. Borenstein and Y. Koren, "Obstacle avoidance with ultrasonic sensors," *IEEE Journal of Robotics and Automation*, vol. 4, no. 2, pp. 213–218, 1988.

[9] J. L. Crowley, "World modeling and position estimation for a mobile robot using ultrasonic ranging," in *, 1989 IEEE International Conference on Robotics and Automation, 1989. Proceedings*, 1989, pp. 674–680 vol.2.

[10] R. Kuc and B. Barshan, "Navigating vehicles through an unstructured environment with sonar," in *, 1989 IEEE International Conference on Robotics and Automation, 1989. Proceedings*, 1989, pp. 1422–1426 vol.3.

[11] V. Lumelsky and T. Skewis, "A paradigm for incorporating vision in the robot navigation function," in *, 1988 IEEE International Conference on Robotics and Automation, 1988. Proceedings*, 1988, pp. 734–739 vol.2.

[12] A. Elfes, "Sonar-based real-world mapping and navigation," *IEEE Journal of Robotics and Automation*, vol. 3, no. 3, pp. 249–265, 1987.

[13] H. P. Moravec and A. Elfes, “High resolution maps from wide angle sonar,” in *1985 IEEE International Conference on Robotics and Automation. Proceedings*, 1985, vol. 2, pp. 116–121.

[14] H. Moravec, “Sensor fusion in certainty grids for mobile robots,” *AI Mag.*, vol. 9, no. 2, pp. 61–74, Jul. 1988.

[15] I. Ulrich and J. Borenstein, “VFH+: reliable obstacle avoidance for fast mobile robots,” in *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, 1998, vol. 2, pp. 1572 –1577 vol.2.

[16] I. Ulrich and J. Borenstein, “VFH*: Local obstacle avoidance with look-ahead verification,” in *Robotics and Automation, 2000. Proceedings. ICRA’00. IEEE International Conference on*, 2000, vol. 3, pp. 2505–2511.

[17] P. Fiorini and Z. Shiller, “Motion Planning in Dynamic Environments Using Velocity Obstacles,” *The International Journal of Robotics Research*, vol. 17, no. 7, pp. 760–772, Jul. 1998.

[18] J. Snape, J. van den Berg, S. J. Guy, and D. Manocha, “Independent navigation of multiple mobile robots with hybrid reciprocal velocity obstacles,” in *Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems*, Piscataway, NJ, USA, 2009, pp. 5917–5922.

[19] E. Prassler, J. Scholz, and P. Fiorini, “Navigating a Robotic Wheelchair in a Railway Station during Rush Hour,” *The International Journal of Robotics Research*, vol. 18, no. 7, pp. 711–727, Jul. 1999.

[20] P. Fiorini and D. Botturi, “Introducing service robotics to the pharmaceutical industry,” *Intel Serv Robotics*, vol. 1, no. 4, pp. 267–280, Oct. 2008.

[21] J. K. Kuchar and L. C. Yang, “A review of conflict detection and resolution modeling methods,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 1, no. 4, pp. 179–189, 2000.

[22] Z. Shiller, R. Prasanna, and J. Salinger, “A Unified Approach to Forward and Lane-Change Collision Warning for Driver Assistance and Situational Awareness,” SAE International, Warrendale, PA, SAE Technical Paper 2008-01-0204, Apr. 2008.

[23] D. Helbing, I. Farkas, and T. Vicsek, “Simulating dynamical features of escape panic,” *Nature*, vol. 407, no. 6803, pp. 487–490, Sep. 2000.

[24] C. W. Reynolds, “Flocks, herds and schools: A distributed behavioral model,” in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, 1987, pp. 25–34.

[25] J. van den Berg, M. C. Lin, and D. Manocha, “Reciprocal Velocity Obstacles for Real-Time Multi-Agent Navigation,” *IEEE International Conference on Robotics and Automation,2008. Proceedings*, 2008, pp. 1928–1935.

[26] Y. Abe and M. Yoshiki, “Collision avoidance method for multiple autonomous mobile agents by implicit cooperation,” in *2001 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2001. Proceedings*, 2001, vol. 3, pp. 1207–1212 vol.3.

[27] B. Kluge and E. Prassler, “Recursive Probabilistic Velocity Obstacles for Reflective Navigation,” in *Field and Service Robotics*, S. Yuta, H. Asama, E. Prassler, T. Tsubouchi, and S. Thrun, Eds. Springer Berlin Heidelberg, 2006, pp. 71–79.

[28] O. Gal, Z. Shiller, and E. Rimon, “Efficient and safe on-line motion planning in dynamic environments,” in *IEEE International Conference on Robotics and Automation, 2009. ICRA ’09*, 2009, pp. 88–93.

[29] J. Snape, J. van den Berg, S. J. Guy, and D. Manocha, “The Hybrid Reciprocal Velocity Obstacle,” *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 696–706, 2011.

[30] K. Okada, M. Kojima, S. Tokutsu, T. Maki, Y. Mori, and M. Inaba, “Multi-cue 3D object recognition in knowledge-based vision-guided humanoid robot system,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2007. IROS 2007*, 2007, pp. 3217–3222.

[31] J. F. G. de Freitas, M. Niranjan, A. H. Gee, and A. Doucet, “Sequential Monte Carlo Methods to Train Neural Network Models,” *Neural Computation*, vol. 12, no. 4, pp. 955–993, Apr. 2000.

[32] C. Hue, J.-P. Le Cadre, and P. Perez, “Sequential Monte Carlo methods for multiple target tracking and data fusion,” *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 309–325, 2002.

[33] A. Doucet, N. de Freitas, and N. Gordon, “An Introduction to Sequential Monte Carlo Methods,” in *Sequential Monte Carlo Methods in Practice*, A. Doucet, N. de Freitas, and N. Gordon, Eds. Springer New York, 2001, pp. 3–14.

[34] K. Okuma, A. Taleghani, N. de Freitas, J. J. Little, and D. G. Lowe, “A Boosted Particle Filter: Multitarget Detection and Tracking,” in *Computer Vision - ECCV 2004*, T. Pajdla and J. Matas, Eds. Springer Berlin Heidelberg, 2004, pp. 28–39.

[35] C. Yang, R. Duraiswami, and L. Davis, “Fast multiple object tracking via a hierarchical particle filter,” in *Tenth IEEE International Conference on Computer Vision, 2005. ICCV 2005*, 2005, vol. 1, pp. 212–219 Vol. 1.

[36] H. Zhou and S. Sakane, “Sensor Planning for Mobile Robot Localization -A hierarchical approach using Bayesian network and particle filter-,” in *IEEE International Conference on Robotics and Biomimetics, 2004. ROBIO 2004*, 2004, pp. 540–545.

[37] D. Schulz, W. Burgard, D. Fox, and A. B. Cremers, “People Tracking with Mobile Robots Using Sample-Based Joint Probabilistic Data Association Filters,” *The International Journal of Robotics Research*, vol. 22, no. 2, pp. 99–116, Feb. 2003.
[41] [38] M.de Berg, O.Cheong, M. van Kreveld, M.Overmars.“Computational Geometry,” in *Computational Geometry*, Springer Berlin Heidelberg, 2008, pp. 1–17.

[38] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, “A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking,” *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 174–188, 2002.

[39] C. S. Agate and K. J. Sullivan, “Particle filtering algorithm for tracking multiple road-constrained targets,” pp. 256–266, Aug. 2003.

[40] S.-C. Chu, J. F. Roddick, and J. S. Pan, “An Efficient K -Medoids-Based Algorithm Using Previous Medoid Index, Triangular Inequality Elimination Criteria, and Partial Distance Search,” in *Proceedings of the 4th International Conference on Data Warehousing and Knowledge Discovery*, London, UK, UK, 2002, pp. 63–72.

[41] Omohundro, Stephen Malvern. *Five balltree construction algorithms*. Berkeley: International Computer Science Institute, 1989.

[42]K. L. Clarkson, “Fast algorithms for the all nearest neighbors problem,” in *, 24th Annual Symposium on Foundations of Computer Science, 1983*, 1983, pp. 226–232.