SIMULATING COLLISION DETECTION AND AVOIDANCE IN HIGHLY DYNAMIC ENVIRONMENT

Nishad Pancholi
B.E., North Gujarat University, India, 2007

PROJECT

Submitted in partial satisfaction of the requirements for the degree in

MASTER OF SCIENCE

in

ELECTRICAL AND ELECTRONIC ENGINEERING

at

CALIFORNIA STATE UNIVERSITY, SACRAMENTO

SPRING
2011
SIMULATING COLLISION DETECTION AND AVOIDANCE IN HIGHLY DYNAMIC ENVIRONMENT

A Project

by

Nishad Pancholi

Approved by:

_____________________________, Committee Chair
Fethi Belkhouche, Ph.D.

_____________________________, Second Reader
Preetham B. Kumar, Ph.D.

__________________________________
Date
Student: Nishad Pancholi

I certify that this student has met the requirements for format contained in the university format manual and that this project is suitable for shelving in the library and credits to be rewarded for the Project.

__________________________, Graduate Coordinator       ________________
Preetham B. Kumar, Ph.D.            Date

Department of Electrical and Electronic Engineering
Abstract

of

SIMULATING COLLISION DETECTION AND AVOIDANCE IN HIGHLY DYNAMIC ENVIRONMENT

by

Nishad Pancholi

This project develops methods for collision detection between multiple robots in highly dynamic environment. Kinematics based characterization is proposed to precisely identify collision. Collision avoidance algorithm developed using priority of the robot. Based on the priority of the robot, change in the velocity is proposed to avoid collision. Simulation of this algorithm is performed using MATLAB. In simulation initially collision detection and avoidance between two robots are proposed. Later this approach is extended to detect collision between multiple moving robots. Highly dynamic environment is created for the simulation where these algorithms running in parallel for each robot. Change in the programming code can be made to create different types of environment.

_______________________, Committee Chair
Fethi Belkhouche, Ph.D.

_______________________Date

iv
I would like to take this opportunity to thank the people behind successful completion of this project. First and foremost, I would like to thank my professor and mentor for this project, Dr. Fethi Belkhouche for his tremendous support and guidance throughout the project. I would like to thank him for always being there to help me in developing better understanding of the topic and encouraging me to give my best. Also, I am extremely grateful to my Graduate Coordinator, Dr. Preetham Kumar for his consistent support in my academic years at the School guiding me through the entire process, right from enrolling for a class until submission of my master’s project. Finally, I would like to thank my professors throughout my Master’s, my department and my school for giving me this opportunity to learn and grow towards attainment of this degree.

Lastly, I would like to express my heartfelt gratitude to my family in India who have constantly supported me and blessed me. I am extremely thankful to my fiancé Dimple, who helped me make my dreams come true.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgements</td>
<td>v</td>
</tr>
<tr>
<td>List of Figures</td>
<td>vi</td>
</tr>
<tr>
<td>Chapter</td>
<td></td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2. ALGORITHMS FOR COLLISION DETECTION AND AVOIDANCE</td>
<td>4</td>
</tr>
<tr>
<td>3. APPROACH</td>
<td>18</td>
</tr>
<tr>
<td>3.1 Initial Approach - Collision Between Two Robots</td>
<td>18</td>
</tr>
<tr>
<td>3.2 Final Approach - Collision Between Multiple Robots</td>
<td>27</td>
</tr>
<tr>
<td>4. SIMULATION TECHNIQUES AND TOOLS</td>
<td>30</td>
</tr>
<tr>
<td>5. CONCLUSION</td>
<td>36</td>
</tr>
<tr>
<td>Appendix MATLAB Code</td>
<td>37</td>
</tr>
<tr>
<td>References</td>
<td>48</td>
</tr>
</tbody>
</table>
**LIST OF FIGURES**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Geometry of robot and moving object in 2-D plane</td>
<td>9</td>
</tr>
<tr>
<td>2.</td>
<td>Robot and moving object are approaching from each other but collision does not take place</td>
<td>11</td>
</tr>
<tr>
<td>3.</td>
<td>Robots moving away from each other</td>
<td>21</td>
</tr>
<tr>
<td>4.</td>
<td>Robots colliding</td>
<td>22</td>
</tr>
<tr>
<td>5.</td>
<td>Line of sight and positive error angle E</td>
<td>23</td>
</tr>
<tr>
<td>6.</td>
<td>Line of sight and negative error Angle E</td>
<td>24</td>
</tr>
<tr>
<td>7.</td>
<td>Definition of angle E</td>
<td>25</td>
</tr>
<tr>
<td>8.</td>
<td>Pairs of robots with local boundaries in dynamic environment</td>
<td>27</td>
</tr>
<tr>
<td>9.</td>
<td>Robots in highly dynamic environment</td>
<td>28</td>
</tr>
<tr>
<td>10.</td>
<td>Flow chart for the collision detection between two robots</td>
<td>32</td>
</tr>
<tr>
<td>11.</td>
<td>Flow chart for the collision avoidance between two robots</td>
<td>33</td>
</tr>
<tr>
<td>12.</td>
<td>Simulation Graph for two robots</td>
<td>34</td>
</tr>
<tr>
<td>13.</td>
<td>LOS angles for different iterations</td>
<td>34</td>
</tr>
<tr>
<td>14.</td>
<td>Initial stage of simulation for dynamic environment</td>
<td>35</td>
</tr>
<tr>
<td>15.</td>
<td>Intermediate stage of simulation for dynamic environment</td>
<td>35</td>
</tr>
</tbody>
</table>
Chapter 1

INTRODUCTION

When a robot operates in an unstructured environment or shares its workspace with other robots, structures or human users, safety issues are of primary concern. Injuries may occur from an accidental collision between the robot and the environment (i.e. other robots), due to the uncertain location of obstacles and/or unpredicted relative motion. Avoiding such collisions requires knowledge of the environment geometry and the motion of other moving robots.

The ability to avoid collision with moving obstacles is necessary for various applications of mobile robots. This ability significantly increases the mobility of a robot for navigation and allows robot to perform other tasks. Planning collision free paths in dynamic environment requires a dynamic collision avoidance system. In dynamic environment path planning is more difficult. In general, collision detection deals with the problem of checking whether two objects overlap in space or whether their boundaries intersect as they move. In different fields collision detection problems can be seen from different point of views. For example, probabilistic approaches are widely used in aerospace engineering while these techniques are not used in computer graphics. Collision detection is very important to get immersive performances for robots moving in 3-Dimensional objects in the virtual reality system. The algorithms of collision detection are different depending on the applications.
The general idea behind collision detection is to detect whether different objects or moving robots are going to collide with each other in future time. Taking time into account, collision detection can be divided into continuous collision detection and discrete collision detection. Collision detection with stationary objects can be performed easily using today’s technology. Collision detection for the moving objects is more difficult. Over the last decade, there have been important developments of different algorithms for the different applications.

The following points should be considered for collision detection between moving robots.

1. In the dynamic environment, there are many robots and objects moving. The motion of the objects and other robot is not necessarily known.

2. It requires to judge whether the distance between two robots is equal or less than the minimum safety distance.

In the following report, chapter 2 gives information on different collision detection algorithms. It also includes algorithm to detect collision for this project. Further, this chapter gives information on different collision avoidance techniques. It also includes algorithms to avoid collision. Chapter 3 initially gives information on initial approach of simulation of collision detection and avoidance of two robots. Then it extends the discussion for final approach of simulation of collision detection and avoidance of multiple robots in highly dynamic environment. Chapter 4 gives information on
Simulation techniques and tools used for this project. Chapter 5 gives summary of the project as well as scope for future expansion. In the Appendix section a detailed MATLAB code is given. Lastly, in the Reference section references used for this report as well as this project, is given.
There are many collision detection methods developed for different applications. The following is the brief introduction for some of those methods.

**Space Decomposition Method:** The principal of space decomposition is to break down the virtual space into many equal volume grids and tests objects inhibited in the same grid or sideward grids. Typical methods include Octree method and Binary Space Partitioning (BSP). The principle to the Space Decomposition is that the two objects can not overlap, if they are in different grids. Further, overlapping test is needed if two objects are located in the same grid.

**Bounding Volume Hierarchical:** The principal of bounding volume hierarchical is to wrap the complicated geometrical objects with big volume and simple shape boxes. For collision detection, we should do overlapping tests among bounding boxes at first. Then, geometrical objects should be detected accurately. Bounding volume method is effective to judge whether two geometrical objects are overlapping. There are many types of bounding volumes: bounding Spheres, axis-aligned bounding boxes (AABB), oriented bounding boxes (OBB), discrete orientation polyhedrons (k-DOP), and many more.
For space decomposition methods, collision detection efficiency is low, when the moving objects are located very close to each other. For the bounding volume hierarchical method, the moving speed of the robots or objects should be limited and the motion track should be simple. Moreover, it is quite difficult to set up models for this method, especially for complicated and sudden motion change.

There is no algorithm which can be applied to all cases. Most of collision detection algorithms are designed for idiographic applications. Furthermore, there are few high-effective detection algorithms for the dynamic and complex virtual environment.

In general, collision detection includes two phases. First phase is pre-collision detection. One should start taking actions when the distance between any moving objects or robots is less than the minimum safety distance. For this purpose, radar ranging or other sensors method can be used to predict a potential area of collision detection [1]. Different types of sensors including vision sensors can be used for this first step depending on the application.

Second phase is accurate collision detection. The accurate collision detection should be started immediately after pre-collision detection. In this phase, the possible collision is determined according to the robots’ relative direction angle [1].
Besides the above techniques neural networks can also be used for real time collision detection. Adaptive control law was also suggested for the dynamic collision detection.

Following is the description of the algorithm of collision detection for this project. Collision course between the robot and moving object means that, given the instantaneous motion profile and the positions of the robot and the moving object, a collision will take place in the future. For this project characterization of the collision course is kinematics based. The precise goal for this algorithm is to predict exact collision course i.e. whether the robot will in near future collide or not, with other moving objects or robots.

The robot and moving object are moving in 2-dimensional horizontal plane, where we use the relative kinematics equations between the robot and the moving object. The conditions for the collision course are derived in terms of the angular and linear velocities, and the relative velocities. The robot is modeled as a geometric point.

Let's consider a 2-dimensional global reference space $S$. The space $S$ is not infinite space but it has boundary on X and Y axis. This boundary can be different for different problems. Let $R$ denote a mobile robot moving in a 2-dimensional workspace $S$. This robot has velocity $V_R$. We assume that the dynamics of the robot can be described by a
differential equation of the form $\dot{s}_r = f(s_r, v_r)$, where $s_r$ belongs to $S$, is the state of robot and $v_r$ is the velocity of the robot.

Let $B$ denote an object moving in the workspace $S$. The motion of $B$ is characterized by a differential equation such as $\dot{s}_b = f(s_b, v_b)$, where $s_b$ belongs to $S$, is the state of moving object and $v_b$ is the velocity of the object.

Besides information given about the motion dynamics of the robot and the moving object, we assume that:

1. The robot does not have any prior knowledge of the motion of the moving object. However, the moving object's velocity and position are assumed to be measurable in real time by the robot’s sensory system.

2. Both the robot and the moving object have circular shapes but in order to simplify the analysis, the robot is reduced to its center point. We can later enlarge them to a particular radius. This approach is widely used in robot navigation.

3. The moving object is rigid body, where all its points move at the same velocity. We are also ignoring the movement of the robot with reference to its own center, i.e. we are considering holonomic motion only.

Based on the geometry of the robot and the object described in figure 1, the equations of system 1 and system 2 can be obtained for 2-dimensional space $S$. 
For the robot,

\[ X_R = V_R \cos \theta_R \]
\[ Y_R = V_R \sin \theta_R \]  \hspace{1cm} (1)
\[ \theta_R = W_R \]

Based on the equations of system (1), the robot's state is given by \( \hat{\mathbf{s}}_r = [X_R, Y_R, \theta_R, V_R] \), where \( V_R \) is the robot's linear velocity, \( \theta_R \) is the robot's orientation angle and \( W_R \) is the robot's angular velocity. \( (X_R, Y_R) \) represents the robot's position in the global reference plane.

For the moving object,

\[ X_B = V_B \cos \theta_B \]
\[ Y_B = V_B \sin \theta_B \]  \hspace{1cm} (2)
\[ \theta_B = W_B \]
Based on system (2), the moving object's state is given by $\hat{s}_b = [X_B, Y_B, \theta_B, V_B]$, where $V_B$ is the object's linear velocity, $\theta_B$ is the object's orientation angle and $W_B$ is the object's angular velocity. $(X_B, Y_B)$ represents the moving object's position in the global reference plane.
As shown in figure 1, consider an imaginary line starting from the robot to the moving object or vice versa. This imaginary line is known as the line of sight and it is abbreviated as LOS. The distance of this LOS is the relative distance between robot R and object B, and can be defined as,

\[ D_{BR} = \sqrt{(X_R - X_B)^2 + (Y_R - Y_B)^2} \]  \hspace{1cm} (3)

Also, consider an angle from the positive x-axis to the line of sight. This angle is also referred as line of sight of angle. The expression for this sight of angle \( \lambda_{BR} \) is,

\[ \tan \lambda_{BR} = \frac{Y_R - Y_B}{X_R - X_B} \]

or, \( \lambda_{BR} = \tan^{-1} \frac{Y_R - Y_B}{X_R - X_B} \)  \hspace{1cm} (4)

Consider the velocity vector,

\[ \vec{V}_{BR} = \vec{V}_B - \vec{V}_R \]  \hspace{1cm} (5)

The relative velocity can be decomposed into two components in the Cartesian plane as follows,

\[ \vec{V}_{BR} = V_{BRx} \vec{U}_x + V_{Bry} \vec{U}_y \]  \hspace{1cm} (6)

Where \( \vec{U}_x \) and \( \vec{U}_y \) are the unit vector along the x and y axes, respectively. By using the robots’ and moving objects’ kinematics equations, we get

\[ V_{BRx} = X_D = V_B \cos \theta_B - V_R \cos \theta_R \]

\[ V_{Bry} = Y_D = V_B \sin \theta_B - V_R \sin \theta_R \]  \hspace{1cm} (7)

With \( X_D = X_B - X_R \), \( Y_D = Y_B - Y_R \), system (7) provides complete description of
the motion of the moving object as seen by the robot. However, this system is highly nonlinear and difficult to solve analytically.

Generally two objects are in collision course if they are approaching from each other. However, this condition can lead to a wrong conclusion about collision.

![Diagram](image-url)

Figure 2: Robot and moving object are approaching from each other but collision does not take place
Consider the scenario in figure 2, where the robot and the moving object are moving at
the same speed. The robot and the moving object are approaching from each other in the
time intervals shown in the figure 2, but they are not in a collision course, since the robot
reaches the path of intersection point before the moving object.

Our aim is to characterize exact collision course based on the relative kinematics
equations. This will allow the prediction of future collision course based on the states of
the robot and the moving object. These states are characterized by $V_R$, $V_B$, $\theta_R$, $\theta_B$, $\lambda_{BR}$.

From the above discussion we can conclude that paths intersection is a necessary
condition for collision. However, path intersection does not imply collision all the time.
Next, we discuss the paths intersection by using the instantaneous values of the
orientation angles of the robot and the moving object. We have the following result
concerning geometric paths intersection.

Proposition 1: For $V_B > 0$, $V_R > 0$, geometric paths intersection can be characterized by
the following conditions,

$$ C = \frac{\sin(\theta_R - \theta_B)}{\sin(\theta_R - \lambda_{BR})} < 0 $$

and,

$$ D = \frac{\sin(\theta_R - \theta_B)}{\sin(\theta_B - \lambda_{BR})} < 0 $$

From (8) and (9), the sign of $\sin(\theta_R - \lambda_{BR})$ and $\sin(\theta_B - \lambda_{BR})$ is the same when the paths
intersect [1].
Note that (8) and (9) are valid for constant and time varying orientation angles and line of sight angle. The robot and the moving object are in a collision course if they will arrive at the paths intersection point P at the same time. Now we discuss the collision course based on the relative kinematics equations.

Our main result is stated as follows.

Proposition: Let $K$ be a constant. If (8) and (9) are satisfied for $V_R > 0$, $V_B > 0$, with

$$\frac{X_D}{Y_D} = K$$

Then, the robot and the moving objects are in collision course, which means they will reach point P at the same time. It can be shown by taking the time derivative in equation (4),

$$K = \tan \lambda_{BR}$$

$K$ is constant. Therefore equation (10) can be written as,

$$Y_D = X_D \tan \lambda_{BR}$$

If $\tan \lambda_{BR}$ is constant $\lambda_{BR}$ is also constant and therefore the time rate of $\lambda_{BR}$ is zero.

The relative distance between the robot and the moving object projected on the y-axis is proportional to the relative distance projected to the relative distance projected on x-axis, with a constant proportionality factor. The proof for proposition 2 can be sated as follows:

The proof of the above proposition is based on the relative range $r_{BR}$.

Transformation to polar coordinates of systems (1) and (2) yields the following system
\[
\dot{r}_R = V_R \cos(\theta_R - \lambda_R)
\]
\[
r_R \dot{\lambda}_R = V_R \sin(\theta_R - \lambda_R)
\]
\[
\dot{\theta}_R = W_R
\]

For the robot, where \((r_R, \lambda_R)\) are the robot’s coordinates, and

\[
\dot{r}_B = V_B \cos(\theta_B - \lambda_B)
\]
\[
r_B \dot{\lambda}_B = V_B \sin(\theta_B - \lambda_B)
\]

For the moving object \((r_B, \lambda_B)\) are the polar coordinates. Similarly, according to (6) the relative velocity can also be decomposed into two components along and across the LOS as follows,

\[
\vec{V}_{BR} = V_{\perp}^{\perp} \vec{U}_\perp + V_{\parallel}^{\parallel} \vec{U}_\parallel
\]

where \(\vec{U}_\perp\) and \(\vec{U}_\parallel\) are the unit vectors across and along the LOS respectively.

By considering systems (13) and (14), we get the relative velocity components

\[
\dot{r}_{BR} = V_B \cos(\theta_B - \lambda_{BR}) - V_R \cos(\theta_R - \lambda_{BR})
\]
\[
r_{BR} \dot{\lambda}_{BR} = V_B \sin(\theta_B - \lambda_{BR}) - V_R \sin(\theta_R - \lambda_{BR})
\]

Recall that (8) and (9) result from the paths intersection conditions. From equation (10), since \(\tan(\lambda_{BR}) = \text{constant}\), it turns out that the rate of change of \(\lambda_{BR}\) is zero. This gives,

\[
V_R \sin(\theta_R - \lambda_{BR}) = V_B \sin(\theta_B - \lambda_{BR})
\]

By using equation (17) and the equation for the relative velocity between the robot and the moving object, we get

\[
\dot{r}_{BR} = V_R \frac{\sin(\theta_R - \theta_B)}{\sin(\theta_B - \lambda_{BR})} \cos(\theta_R - \lambda_{BR}) - V_R \cos(\theta_R - \lambda_{BR})
\]
By using geometric identities we get in terms of robot moving object range.

\[
\dot{r}_{BR} = V_R \frac{\sin(\theta R - \theta B)}{\sin(\theta B - \lambda BR)} = C_{vR}
\]  \hspace{1cm} (19)

Or in terms of velocity moving object \( V_B \)

\[
\dot{r}_{BR} = V_R \frac{\sin(\theta R - \theta B)}{\sin(\theta B - \lambda BR)} = C_{vB}
\]  \hspace{1cm} (20)

Once we know that the robot and the moving object are going to collide, our next aim is to avoid this collision. It should be noted that above kinematics based characterization is for exact collision detection. So, unlike pre-collision detection phase, it is compulsory to avoid detected collision immediately.

For many projects, the moving paths of the moving robots are designed such that the robots will not collide with each other or with moving objects. In other words, paths are predefined. These projects are different from the prospective of our project. Collision is avoided simply because all the robots have predefined paths. While in our project, none of the robot has predefined path in 2-dimensional system. So, for our project continuous checking of collision detection and avoidance is compulsory.

To avoid the collision, change of parameters of either one robot or both robots is necessary. Based on the requirement of the project following collision avoidance algorithm is developed.
After detecting collision between two robots, change of the velocity of the robots is very effective solution to avoid the collision. In the first scenario, we can change the velocity of the first robot. In the second scenario, we can change the velocity of the second robot. Changing the velocity of both robots by same value, does not solve the problem. One should be careful for this condition, while applying this collision avoidance algorithm.

In our project, individual robot does not have definite path but it has definite start and end point in the 2-dimentional plane. Because environment is highly dynamic, we are considering that each robot has a priority amongst the whole group of robots in the environment. This global priority is defined in such a way that, if the first robot has higher priority than the second robot, first robot should not have to slow down. Considering priority as a parameter for changing the velocity of robot, we can conclude the following statements for our algorithm.

1) If robot A has higher priority than robot B and both robots are in the collision course, robot B has to decrease its velocity or robot A has to increase its velocity.

2) If robot B has higher priority than robot A and the robots are in the collision course, robot A has to decrease its velocity or robot B has to increase its velocity.
Interesting scenario occurs when both the robots have same priority. To handle this situation, our algorithm has to consider the angle of direction of the robot. We can extend our algorithm to the following statement.

3) When robot A and robot B have same priority and both the robots are in the collision course, either robot A or robot B has to change its angle of direction.

Based on the three statements of collision avoidance algorithm, the programming flow chart is made for the simulation. The details of this chart and simulation techniques are described in chapter 4.
3.1 Initial Approach - Collision Between Two Robots

For this project we need to create highly dynamic environment with robots and moving objects. So far for the collision detection and avoidance, we are considering that both robot and moving object have velocity, particular shape and angle of direction.

Now for the simplicity, instead of robots and moving object, consider only multiple robots with different velocity and different angles of direction. And we are also assuming circular shape for all robots.

Instead of directly working on the highly dynamic environment, we first focus on the two robots problem. This approach for collision detection and avoidance is based on the algorithm discussed in the chapter 3. Based on this initial approach we can derive the problem for the highly dynamic environment. Further, complexity of dynamic environment cannot be an overhead, once we are considering enough situations for the collision detection and avoidance of the two robots.
Problem description:

Consider a 2-dimensional system, in which two robots are moving. Consider one robot has radius $R_1$ velocity $V_1$ and angle of direction $\theta_1$. The other robot has radius $R_2$ velocity $V_2$ and angle of direction $\theta_2$.

At time $t = t_0$, we can find the X and Y co-ordinates for these two robots. The sampling time is denoted by $h$. After sampling, velocity and angle of direction, we can write the following equations for current co-ordinates of the centers of the round shaped robots.

For robot 1,

$$X_1 = h \ast V_1 \ast \cos\theta_1 + X_1(\text{old})$$

$$Y_1 = h \ast V_1 \ast \sin\theta_1 + Y_1(\text{old}) \quad (21)$$

For robot 2,

$$X_2 = h \ast V_2 \ast \cos\theta_2 + X_2(\text{old})$$

$$Y_2 = h \ast V_2 \ast \sin\theta_2 + Y_2(\text{old}) \quad (22)$$

Here, iteration $h$ is same for both robots for the current time slot. $X_1$, $Y_1$, $V_1$, $\theta_1$ are the X-coordinate, Y-coordinate, Velocity and angle of direction of the first robot respectively. It is important to mention that X and Y coordinates are reference co-ordinates. Their reference is the previous value of the X and Y. So, $X_1(\text{old})$ and $Y_1(\text{old})$ are added in the reference position of the new $X_1$ and new $Y_1$ to get the exact location of current $X_1$ and current $Y_1$ in the 2-dimensional system.
Likewise, \( X_2, Y_2, V_2, \theta_2 \) are the X-coordinate, Y-coordinate, velocity and angle of direction of the second robot respectively. \( X_2(\text{old}) \) and \( Y_2(\text{old}) \) are added in the reference position of the new \( X_2 \) and new \( Y_2 \) to get the exact location of current \( X_2 \) and current \( Y_2 \) in the two dimensional system.

The distance between these two centers of robot can be defined as distance \( D_1 \) by following equation.

\[
D = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2} \tag{23}
\]

Distance \( D \) is the distance of the line of sight. The angle from the positive x-axis to the line of sight is also known as line of sight angle \( \lambda \). The expression for \( \lambda \) is as follows,

\[
\tan \lambda = \frac{Y_2 - Y_1}{X_2 - X_1}
\]

\[
\lambda = \tan^{-1} \left( \frac{Y_2 - Y_1}{X_2 - X_1} \right) \tag{24}
\]

For the current iteration, the values of \( D \) and \( \lambda \) can be found using equation (23) and (24). The current values of \( D \) and \( \lambda \) can be denoted as \( D_{\text{current}} \) and \( \lambda_{\text{current}} \).

Now for the next iteration, at time \( t = t_1 \), these two robots have new position based on their velocity and angle of direction. At the next iteration, in equation (21) and (22), we can find the new coordinates of the both robots. For this second iteration, the value of the \( X_1(\text{old}), Y_1(\text{old}), X_2(\text{old}) \) and \( Y_2(\text{old}) \) are the values of the \( X_1, Y_1, X_2 \) and \( Y_2 \) of the first iteration respectively. After the new \( X \) and \( Y \) co-ordinates, from equation (23) and (24), \( D \) and \( \lambda \) can be found. The values of \( D \) and \( \lambda \) can be denoted as \( D_{\text{Next}} \) and \( \lambda_{\text{Next}} \).
As the line of sight angle ($\lambda$) is constant, there is a chance for these robots to be in collision course. But this condition is not enough to imply collision between the two robots.

![Figure 3: Robots moving away from each other](image)

From figure (3), it is clear that even if the line of sight is constant for two consecutive iterations, robots are moving away from each other and will not collide in future.
Here it is interesting to look at the condition for the $D_{\text{current}}$ and $D_{\text{Next}}$. For both of the scenarios mentioned in figures (3) and (4), line of sight angle remains constant. But the line of sight distance $D$ is either increasing, in the case robots are moving away from each other, or decreasing in the case of robots are approaching from each other. From this information, we can easily derive two compulsory conditions for detecting collision course of the robots.

Condition (1): The line of sight of angle ($\lambda$) is constant for two consecutive iterations.
\[ \lambda_{\text{Next}} = \lambda_{\text{current}} \]

Condition (2): The line of sight distance (D) is decreasing for two consecutive iterations.

\[ D_{\text{Next}} < D_{\text{current}} \]

Figure 5: Line of sight and positive error angle E

Our original assumption is that the robots are round shaped with some radius. But the above two condition can detect the collision course of the centers of the robots, not the whole robots.
It is interesting to look at the figure (5) and (6). These figures say that even if the line of sight angle ($\lambda$) is not constant (Condition 1 is false) and the distance $D$ tends to be decreasing (Condition 2 is true), the robots are still in collision course.

Thus redefinition of the condition 1 is necessary. The range of the value of the line of sight of angle $\lambda_{\text{Next}}$ should be changed for the comparison. Introduction of the error angle $E$ is necessary here.
From figure (5) and figure (6), it is clear that if line of sight of angle of the next iteration $\lambda_{\text{Next}}$ is within the range from $\lambda_{\text{current}} + \text{angle E}$ to $\lambda_{\text{current}} - \text{angle E}$, there is a chance of collision.

To find angle E consider the figure (7).

![Diagram of angle E](image)

Figure 7: Definition of angle E

It is clear from the figure that,

$$\sin E = \frac{R}{D}$$

$$E = \sin^{-1}\left(\frac{R}{D}\right)$$

(25)

So condition (1) can be rewritten as follows,
New condition (1): The line of sight of angle ($\lambda$) is constant for two consecutive iterations or line of sight of angle of the next iteration ($\lambda_{\text{Next}}$) is within the range from $\lambda_{\text{current}} + \text{Angle E}$ to $\lambda_{\text{current}} - \text{Angle E}$. 

$$\lambda_{\text{current}} - \text{E} < \lambda_{\text{Next}} < \lambda_{\text{current}} + \text{E}$$

If new condition (1) and condition (2) are satisfied, then both robots are in collision course.

Now consider one robot has priority $P_1$ and other robot has priority $P_2$. If $P_1 > P_2$ then we need to change the velocity of the second robot. If $P_2 > P_1$ then we need to change the velocity of the first robot. If $P_1 = P_2$, we can change the angle of direction for any robot.
3.2 Final Approach - Collision Between Multiple Robots

Section 3.1 describes some situations and conditions for collision detection and avoidance between two robots. Generalization of more than two robots is not straightforward. We need a more robust way to develop highly dynamic environment.

![Pairs of robots with local boundaries in dynamic environment](image)

Figure 8: Pairs of robots with local boundaries in dynamic environment
For the first step, to create pairs of two robots in work space. As shown in figure 8, we can consider the moving space of each pair of robots as a subset of global space. Collision detection and avoidance between the robots of each pair is done same way as described in section 3.1.

Figure 9: Robots in highly dynamic environment
After we make sure that we are able to detect and avoid collision very well, in our primary environment, we need to create more complicated dynamic environment. As shown in the figure 9, instead to restricting collision detection and avoidance between two robots, the algorithm is expanded to any surrounding robot. It needs to run collision detection and avoidance algorithm a higher number of times for a single iteration.
Chapter 4

SIMULATION TECHNIQUES AND TOOLS

After deriving algorithms for collision detection and avoidance in highly dynamic environment, we need to create simulator for our project. Actually, in our project development life cycle, first step is to define specification and requirement for the robot’s moving profile and dynamic environment. The next step is to develop robust algorithms based on the specification. The last and most important step of the project is to simulate those algorithms. By simulation we can provide solid proof for those algorithms.

Simulation in the field of robotics is necessary before we start the process of making hardware of the robots. It is too expensive to remove the defects after manufacturing of the whole robot system. By simulation, we can provide different graphs, log files, data files as solid proofs of our algorithm. If we detect the problem in our algorithms during simulations, we may need to redesign the algorithm. We might need several iterations for designing algorithm and simulation. But this iteration time is paid off as we can dramatically reduce the defects, before manufacturing.

It is important to discuss the tool selection to develop the simulation environment. We choose MATLAB tool for both collision detection and avoidance. The selection of this tool is not random. The following are the competitive reasons,
1) MATLAB provides strong library with the very advanced and useful mathematical functions. Built in mathematical functions can dramatically reduce the simulator development time.

2) MATLAB also supports object oriented programming system. The combination of programming system with strong inbuilt mathematical functions is very well suitable for our environment.

3) MATLAB supports graph representation of mathematical functions and variables, used in programming. It also supports movie capturing of graph with defined time frame. This feature is very helpful for verification of the whole simulation process. We can actually see the moving robots, in the form of rounds, based on the velocity and the angle of direction. We can also verify whether the robots are colliding or not.

In the process of development of simulator, first we develop the flow charts for the algorithms described in chapter 3 and 4. Figure 10 and Figure 11 show these flow charts. Based on these flow charts, we develop programming code in MATLAB. This MATLAB code is shown in the Appendix.
Find the X and Y co-ordinates of the centers of the both moving round shaped robot

Find the distance between two centers of robot and store it in D current. Consider this line as LOS

Find the angle between LOS and X-axis. Store it into $\lambda$ current

Find the angle $E$ based on radius of the robot and $D$ current

Calculate $(\lambda \text{ current} + \text{Angle } E) \leq \lambda \text{ Next} \\ \leq (\lambda \text{ current} - \text{Angle } E)$

Yes

Collision Detection = 1

Collision Avoidance Algorithm

Yes

Collision Detection is done

No

Collision Detection is done

Yes

Transfer Local Data to the Internal Storage for next iteration

$\lambda$ Next = $\lambda$ current
$D$ next = $D$ Current

No

Current iteration is done

$D$ current $< D$ Next

Figure 10: Flow chart for the collision detection between two robots
Figure 11: Flow chart for the collision avoidance between two robots

Figure 12 shows simulation graph for the two robots. Figure 13 shows graph for the line of sight angle at each iteration. Figure 12 reveals that two robots are approaching from each other. After first two iterations, the line of sight angle is found constant. Figure 13 also proves that. The priority of the robot moving in X-direction is less than the other robot. Therefore in this scenario, the robot moving in X-direction decreases its velocity and allowing other robot to pass first from the intersection point.
Figure 12: Simulation Graph for two robots

Figure 13: LOS angles for different iterations
Figure 14 shows simulation result of the initial stage for dynamic environment. Figure 15 is the simulation result at some intermediate stage.

Figure 14: Initial stage of simulation for dynamic environment

Figure 15: Intermediate stage of simulation for dynamic environment
Chapter 5

CONCLUSION

This project presents a successful implementation of algorithms for collision detection. The robots are moving in highly dynamic environments. Kinematics principles are used to characterize the collision conditions. Collision between multiple robots at the same time is also taken care of. Collision avoidance algorithm is developed based on the requirements of the project. Priority of the robots is used to change internal parameters of robots to avoid collision. Simulation in MATLAB programming became important factor in verification of the developed algorithms. Graphs from the simulation are obtained as a proof of algorithms. So by changing the some part of the programming code different dynamic environment can be established. Collision avoidance algorithm can also be expanded for the other applications.
File name: Top.M

v1 = 1;
v2 = 1.2;
v3 = 1;
v4 = 1.2;
v5 = 1;
v6 = 1.2;
v7 = 1;
v8 = 1.2;
v9 = 1;
v10 = 1.2;

new_v1 = v1;
new_v2 = v2;
new_v3 = v3;
new_v4 = v4;
new_v5 = v5;
new_v6 = v6;
new_v7 = v7;
new_v8 = v8;
new_v9 = v9;
new_v10 = v10;

priority1 = 10;
priority2 = 1;
priority3 = 11;
priority4 = 1;
priority5 = 1;
priority6 = 11;
priority7 = 1;
priority8 = 1;
priority9 = 10;
priority10 = 1;

h = .5; % Iteration time
theta1 = -pi/2; % Angle of point 1
new_theta1 = theta1;
theta2 = -pi; % Angle of point 2
new_theta2 = theta2;
theta3 = -pi/2; % Angle of point 1
new_theta3 = theta3;
theta4 = -pi; % Angle of point 2
new_theta4 = theta4;

theta5 = -pi/2; % Angle of point 1
new_theta5 = theta5;
theta6 = -pi; % Angle of point 2
new_theta6 = theta6;

theta7 = -pi/2; % Angle of point 1
new_theta7 = theta7;
theta8 = -pi; % Angle of point 2
new_theta8 = theta8;

theta9 = -pi/2; % Angle of point 1
new_theta9 = theta9;
theta10 = -pi; % Angle of point 2
new_theta10 = theta10;

X1=0;Y1=3; % initial position of point 1
X2=3;Y2=0; % initial position of point 2
X3=-45;Y3=45; % initial position of point 1
X4=-40;Y4=40; % initial position of point 2
X5=-35;Y5=35; % initial position of point 1
X6=-30;Y6=30; % initial position of point 2
X7=-25;Y7=25; % initial position of point 1
X8=-20;Y8=20; % initial position of point 2
X9=-15;Y9=15; % initial position of point 1
X10=-10;Y10=10; % initial position of point 2

sigma_first_flag1 = 0;
velocity1_changed = 0;
velocity2_changed = 0;
theta1_changed = 0;
theta2_changed = 0;

sigma_first_flag2 = 0;
velocity3_changed = 0;
velocity4_changed = 0;
theta3_changed = 0;
theta4_changed = 0;

sigma_first_flag3 = 0;
velocity5_changed = 0;
velocity6_changed = 0;
theta5_changed = 0;
theta6_changed = 0;

sigma_first_flag4 = 0;
velocity7_changed = 0;
velocity8_changed = 0;
theta7_changed = 0;
theta8_changed = 0;

sigma_first_flag5 = 0;
velocity9_changed = 0;
velocity10_changed = 0;
theta9_changed = 0;
theta10_changed = 0;

for k=1:h:10 % loop for movement of the point
  %X1 = h*v*cos(theta1)+ X1;
  %Y1 = h*v*sin(theta1)+ Y1;
  [x1,y1,X1,Y1] = motion(new_v1,new_theta1,X1,Y1,h); % X1, Y1 are the new position of the point
  %X2 = h*v*cos(theta2)+ X2;
  %Y2 = h*v*sin(theta2)+ Y2;
  [x2,y2,X2,Y2] = motion(new_v2,new_theta2,X2,Y2,h); % X2, Y2 are the new position of the point
  [x3,y3,X3,Y3] = motion(new_v3,new_theta3,X3,Y3,h);
  [x4,y4,X4,Y4] = motion(new_v4,new_theta4,X4,Y4,h);
  [x5,y5,X5,Y5] = motion(new_v5,new_theta5,X5,Y5,h);
  [x6,y6,X6,Y6] = motion(new_v6,new_theta6,X6,Y6,h);
\[
[x7, y7, X7, Y7] = \text{motion}(new\_v7, new\_theta7, X7, Y7, h);
[x8, y8, X8, Y8] = \text{motion}(new\_v8, new\_theta8, X8, Y8, h);
[x9, y9, X9, Y9] = \text{motion}(new\_v9, new\_theta9, X9, Y9, h);
[x10, y10, X10, Y10] = \text{motion}(new\_v10, new\_theta10, X10, Y10, h);
\]

figure (1)
plot (x1, y1, '*r')
axis ([0 20 0 20])
hold on
plot (x2, y2, '*b')
hold on
plot (x3, y3, '*r')
hold on
plot (x4, y4, '*b')
hold on
plot (x5, y5, '*r')
hold on
plot (x6, y6, '*b')
hold on
plot (x7, y7, '*r')
hold on
plot (x8, y8, '*b')
hold on
plot (x9, y9, '*r')
hold on
plot (x10, y10, '*b')
hold on
pause
figure (2)
plot (x1, y1, '*r')
axis ([0 2 0 2])
hold on
plot (x2, y2, '*b')
hold on
pause

[current_sigma_min1, current_sigma_max1] = \text{sigma}(X1, Y1, X2, Y2); \% \text{atan2}(Y2-Y1, X2-X1);
if (sigma_first_flag1 == 0)
    previous_sigma_min1 = current_sigma_min1 + 1;
else
    current_sigma_min1 = current_sigma_min1 + 1;
end
previous_sigma_max1 = current_sigma_max1 + 1;
sigma_first_flag1 = 1;
end

if(velocity1_changed == 1)
    new_v1 = v1;
    velocity1_changed = 0;
elseif (velocity2_changed == 1)
    new_v2 = v2;
    velocity2_changed = 0;
elseif (theta1_changed == 1)
    new_theta1 = theta1;
    theta1_changed = 0;
elseif (theta2_changed == 1)
    new_theta2 = theta2;
    theta2_changed = 0;
end

collide1 = collision
(current_sigma_min1,current_sigma_max1,previous_sigma_min1,previous_sigma_max1);
if( collide1 == 1 )
    if (priority1 > priority2)
        new_v2 = new_v2/2;
        velocity2_changed = 1;
    elseif (priority2 > priority1)
        new_v1 = new_v1/2;
        velocity1_changed = 1;
    else
        new_theta1 = new_theta1 - 30;
        theta1_changed = 1;
    end
end

previous_sigma_min1 = current_sigma_min1;
previous_sigma_max1 = current_sigma_max1;

[current_sigma_min2,current_sigma_max2] = sigma (X3,Y3,X4,Y4); %atan2 (Y2-Y1, X2-X1);
if (sigma_first_flag2 == 0)
    previous_sigma_min2 = current_sigma_min2 + 1;
    previous_sigma_max2 = current_sigma_max2 + 1;
    sigma_first_flag2 = 1;
end

if(velocity3_changed == 1)
    new_v3 = v3;
    velocity3_changed = 0;
elseif (velocity4_changed == 1)
    new_v4 = v4;
    velocity4_changed = 0;
elseif (theta3_changed == 1)
    new_theta3 = theta3;
    theta3_changed =0;
elseif (theta4_changed == 1)
    new_theta4 = theta4;
    theta4_changed =0;
end

collide2 = collision
  (current_sigma_min2,current_sigma_max2,previous_sigma_min2,previous_sigma_max2);
if( collide2 == 1 )
  if (priority3 >  priority4)
      new_v4 = new_v4/2;
      velocity4_changed = 1;
  elseif (priority4 > priority3)
      new_v3 = new_v3/2;
      velocity3_changed = 1;
  else
      new_theta3 = new_theta3 - 30;
      theta3_changed = 1;
  end
end

previous_sigma_min2 = current_sigma_min2;
previous_sigma_max2 = current_sigma_max2;

[current_sigma_min3,current_sigma_max3] = sigma (X5,Y5,X6,Y6); %atan2 (Y2-Y1, X2-X1);
if (sigma_first_flag3 == 0)
    previous_sigma_min3 = current_sigma_min3 + 1;
    previous_sigma_max3 = current_sigma_max3 + 1;
    sigma_first_flag3 = 1;
end

if(velocity5_changed == 1)
    new_v5 = v5;
    velocity5_changed = 0;
elseif (velocity6_changed == 1)
    new_v6 = v6;
    velocity6_changed = 0;
elseif (theta5_changed == 1)
    new_theta5 = theta1;
    theta5_changed =0;
elseif (theta6_changed == 1)
    new_theta6 = theta6;
    theta6_changed =0;
end

collide3 = collision (current_sigma_min3,current_sigma_max3,previous_sigma_min3,previous_sigma_max3);
if( collide3 == 1 )
    if (priority5 > priority6)
        new_v6 = new_v6/2;
        velocity6_changed = 1;
    elseif (priority6 > priority5)
        new_v5= new_v5/2;
        velocity5_changed = 1;
    else
        new_theta5 = new_theta5 - 30;
        theta5_changed = 1;
    end
end

previous_sigma_min3 = current_sigma_min3;
previous_sigma_max3 = current_sigma_max3;

[c current_sigma_min4, current_sigma_max4] = sigma (X7,Y7,X8,Y8); %atan2 (Y2-Y1, X2-X1);

if (sigma_first_flag4 == 0)
    previous_sigma_min4 = current_sigma_min4 + 1;
    previous_sigma_max4 = current_sigma_max4 + 1;

    sigma_first_flag4 = 1;
end

if(velocity7_changed == 1)
    new_v7 = v7;
    velocity7_changed = 0;
elseif (velocity8_changed == 1)
    new_v8 = v8;
    velocity8_changed = 0;
elseif (theta7_changed == 1)
    new_theta7 = theta7;
    theta7_changed =0;
elseif (theta8_changed == 1)
    new_theta8 = theta8;
    theta8_changed =0;
end

collide4 = collision
(c current_sigma_min4, current_sigma_max4, previous_sigma_min4, previous_sigma_max4);
if( collide4 == 1 )
    if (priority7 > priority8)
        new_v8 = new_v8/2;
        velocity8_changed = 1;
    elseif (priority8 > priority7)
        new_v7 = new_v7/2;
        velocity7_changed = 1;
    else
        new_theta7 = new_theta7 - 30;
        theta7_changed = 1;
    end
previous_sigma_min4 = current_sigma_min4;
previous_sigma_max4 = current_sigma_max4;

[current_sigma_min5, current_sigma_max5] = sigma (X9,Y9,X10,Y10); %atan2 (Y2-Y1, X2-X1);

if (sigma_first_flag5 == 0)
    previous_sigma_min5 = current_sigma_min5 + 1;
    previous_sigma_max5 = current_sigma_max5 + 1;

    sigma_first_flag5 = 1;
end

if(velocity9_changed == 1)
    new_v9 = v9;
    velocity9_changed = 0;

elseif (velocity10_changed == 1)
    new_v10 = v10;
    velocity10_changed = 0;

elseif (theta9_changed == 1)
    new_theta9 = theta9;
    theta9_changed = 0;

elseif (theta10_changed == 1)
    new_theta10 = theta10;
    theta10_changed = 0;
end

collide5 = collision
(current_sigma_min5, current_sigma_max5, previous_sigma_min5, previous_sigma_max5);

if (collide5 == 1 )
    if (priority9 > priority10)
        new_v10 = new_v10/2;
        velocity10_changed = 1;
    elseif (priority10 > priority9)
...
new_v9 = new_v9/2;
velocity9_changed = 1;
else
    new_theta9 = new_theta9 - 30;
    theta9_changed = 1;
end

previous_sigma_min5 = current_sigma_min5;
previous_sigma_max5 = current_sigma_max5;

figure (3)
plot (k,current_sigma_max5,'-+r')
hold on
plot (k,current_sigma_min5,'-+b')
end

File name: Motion.M

function f1=motion(v, theta, xp, yp)
h= 0.1;
xp=0;
yp=0;
for k=1:h:100
    xp = h*v*cos(theta)+ xp;
    yp = h*v*sin(theta)+ yp;
    plot (xp,yp,'-or')
    hold on
end

File name: Lambda.M

function [sigma_min,sigma_max] = sigma (X1,Y1,X2,Y2)
R = .2; % radius of circle
SIGMA = atan2 (Y2-Y1, X2-X1);
r = sqrt (abs(X2-X1)^2 + (abs(Y2-Y1)^2));
e = asin (R/r);
sigma_min = SIGMA - e;
sigma_max = SIGMA + e;
function ret_collision = collision
    (current_sigma_min,current_sigma_max,previous_sigma_min,previous_sigma_max)

    if (previous_sigma_min > previous_sigma_max)
        t = previous_sigma_min;
        previous_sigma_min = previous_sigma_max;
        previous_sigma_max = t;
    end

    if (current_sigma_min > current_sigma_max)
        t = current_sigma_min;
        current_sigma_min = current_sigma_max;
        current_sigma_max = t;
    end

    if ((current_sigma_min >= previous_sigma_min) && (current_sigma_min <=
         previous_sigma_max))
        ret_collision = 1;
    elseif ((current_sigma_max >= previous_sigma_min) && (current_sigma_max <=
                     previous_sigma_max))
        ret_collision = 1;
    elseif ((previous_sigma_min >= current_sigma_min) && (previous_sigma_min <=
                     current_sigma_max))
        ret_collision = 1;
    elseif ((previous_sigma_max >= current_sigma_min) && (previous_sigma_max <=
                     current_sigma_max))
        ret_collision = 1;
    else
        ret_collision = 0;
    end
end
REFERENCES

http://ieeexplore.ieee.org.proxy.lib.csus.edu/search/srchabstract.jsp?tp=&arnumber=4608305&queryText%3Dcollision+detection%26openedRefinements%3D%26searchField%3DSearch+All

[2]. Real-time and Precise Self Collision Detection System for Humanoid Robots  
http://ieeexplore.ieee.org.proxy.lib.csus.edu/search/srchabstract.jsp?tp=&arnumber=1570256&queryText%3Dcollision+detection%26openedRefinements%3D%26searchField%3DSearch+All

[3]. A Scheme Integrating Neural Networks for Real-time Robotic Collision Detection  
http://ieeexplore.ieee.org.proxy.lib.csus.edu/search/srchabstract.jsp?tp=&arnumber=525394&queryText%3Dcollision+detection%26openedRefinements%3D%26searchField%3DSearch+All

[4]. Kinematics-based characterization of the collision course  
F. Belkhouche and B. Belkhouche  
International Journal of Robotics and Automation

[5]. O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots"  