PROJECT REPORT

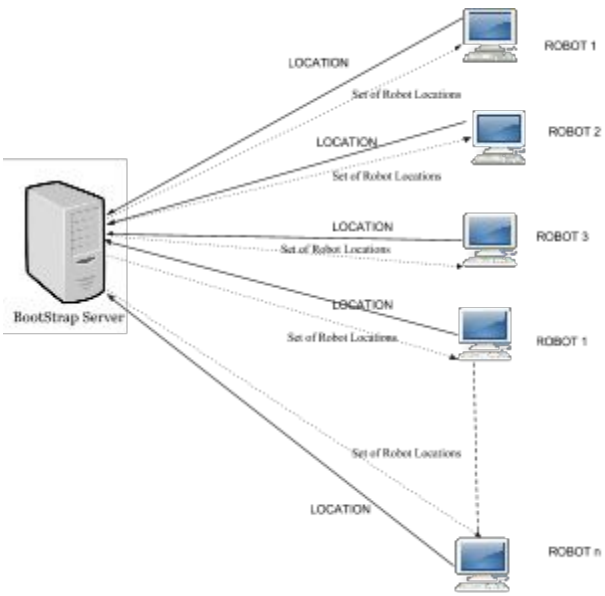
Distributed Robot Attack Simulation

Sagar Shah Gagandeep Malhotra Sumedha Singh

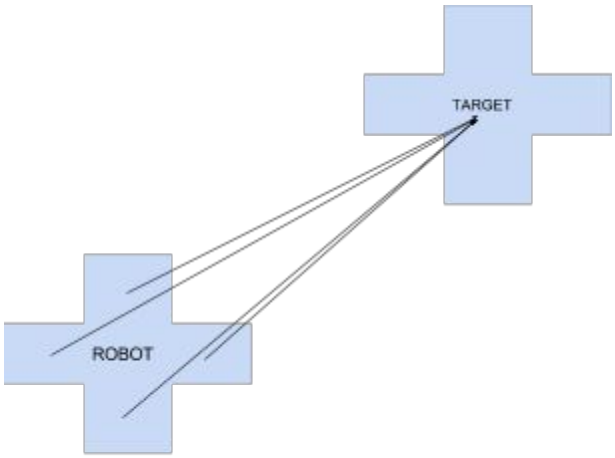
INTRODUCTION

In this project we develop a target tracking system by multiple mobile robot trackers defined by the dynamics of a 2 Dimensional matrix. This a target tracking problem and requires motion strategy among the robots. The aim of the project is to surround the target object in a 2D grid matrix. Each robot is associated with a process that runs on different servers. The robots are communicating directly with each other while the target has no computational capability.

The target will be surrounded by n number of robots using Ricart and Agrawala's mutual exclusion algorithm and coordination algorithm in a distributed environment. The bootstrap server class will have the Board set up and will place the Robots on board and have information of the location of each Robot. One after the other, the Robots will be placed on Board.The Robot Listener will have the list of the positions of all the robots on board. The bootstrap server will send the location of each robot to every robot on board and updates the location information as a new robot is placed on board.

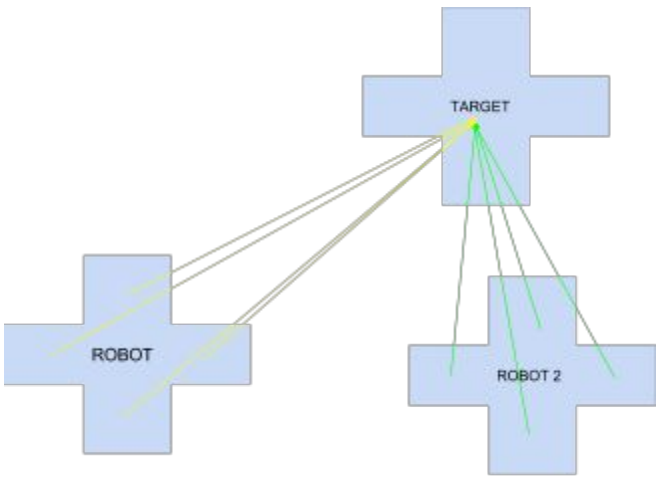


In the first step we will find the next robot location to move to the next cell. The distance of each of the adjoining cell of the robot with that of the target will be calculated using euclidean distance. The cell that will be at the shortest distance from the target cell be the next cell for the robot to move. Similarly, we will calculate this distance for all the robots on board. The cell from which the distance will be shortest from the target will be the next cell to move for the robot.



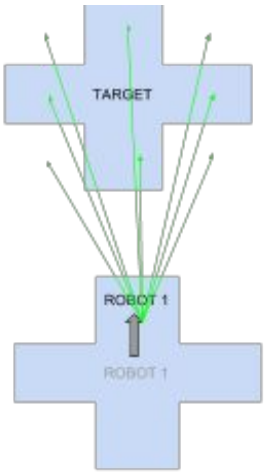
Find the next cell location for each Robot to move

We calculate the euclidean distance from the robot to the target for all the robots in the matrix.

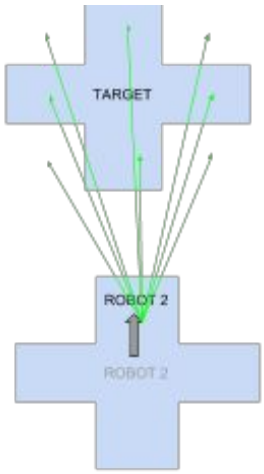


Find the next cell location for all the Robots to move

The robot with the shortest distance from the adjoining robot's cell to the target will be calculated and the next move for each robot will be decided using this distance. Similarly, the robot will calculate the distance after every single move and this will be calculated for all the robots. After the robot has moved one step to its adjoining cell, the next step is to find the distance of the target and its adjoining 8 cells with the robot's new cell location. The robot with the shortest distance will send its target location to Bootstrap Server. Then the next target location for that robot will be fixed and will be send to all other robots.

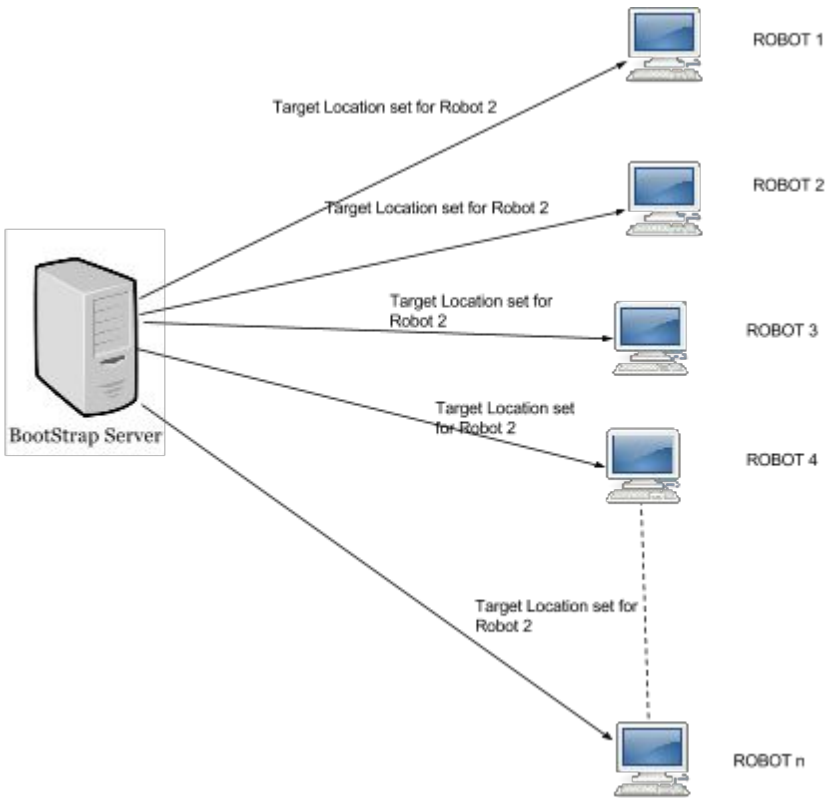


Find the next cell location from the target location for each Robot to move



Find the next cell location from the target location for each Robot to move

The shortest distance calculated using euclidean distance for robot 1 say for example is A. For robot 2 for example the shortest distance is B. Similarly we calculate for all the robots.

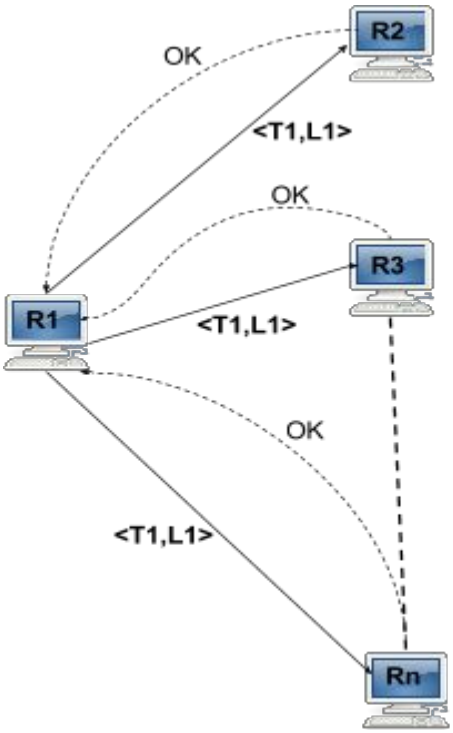


The Server will send the new target location of that robot to all other robots. Now, if say the distance A is shorter than the distance of robot 2 which is B, then robot 1 will set the new location as its target location.

ALGORITHMS AND PROGRAMS

Ricart And Agrawala Algorithm

Ricart and Agrawala's mutual exclusion algorithm is used for coordination between the robots. Mutual exclusion algorithm will work when the robots require to access the critical section. Each Robot will send its Timestamp and Location to all other Robots to request for the critical section. To request the critical section, each Robot will have to make a request to all other Robots. The Robot requesting the critical section will have to get the approval from all other Robots including itself. If any of Robots is not in the critical section, then they can give the approval to that Robot requesting for the CS. Upon receipt from all other Robots, Robot 1 will enter the critical section.



If there are two Robots requesting for the critical section at the same time then their respective timestamps will be compared. For example, if the timestamp of Robot 1 is T1 and of Robot 2 is T2 and T1<T2, then Robot 1 will execute the critical section first and Robot 2 will be send in the queue.

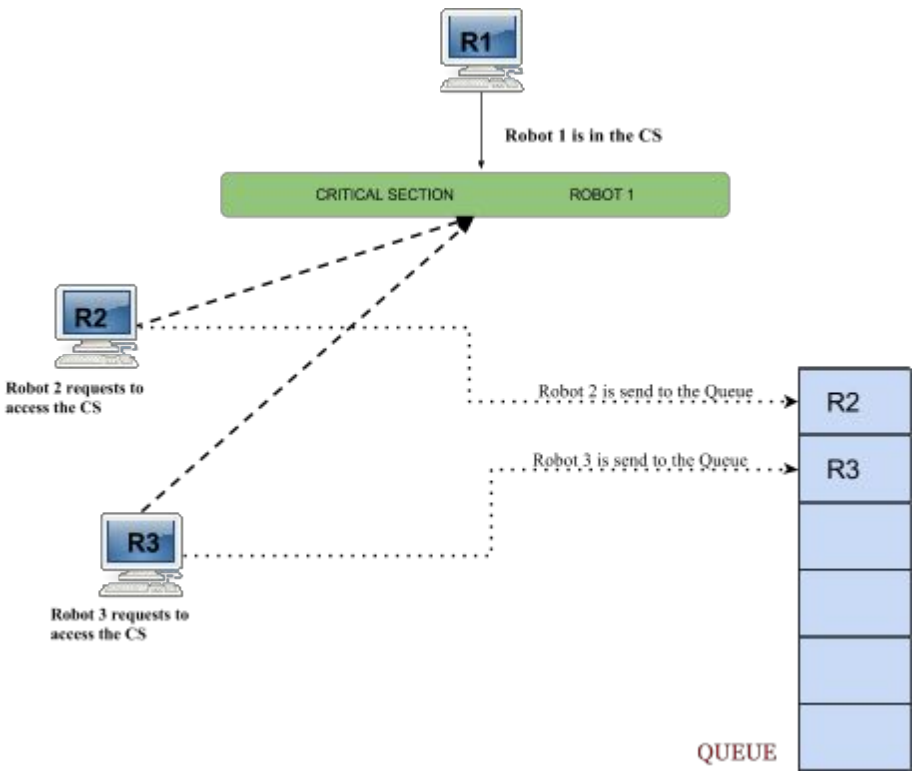
When Robot 1 is in the critical section, then it will buffer the requests from all other Robots in the queue.

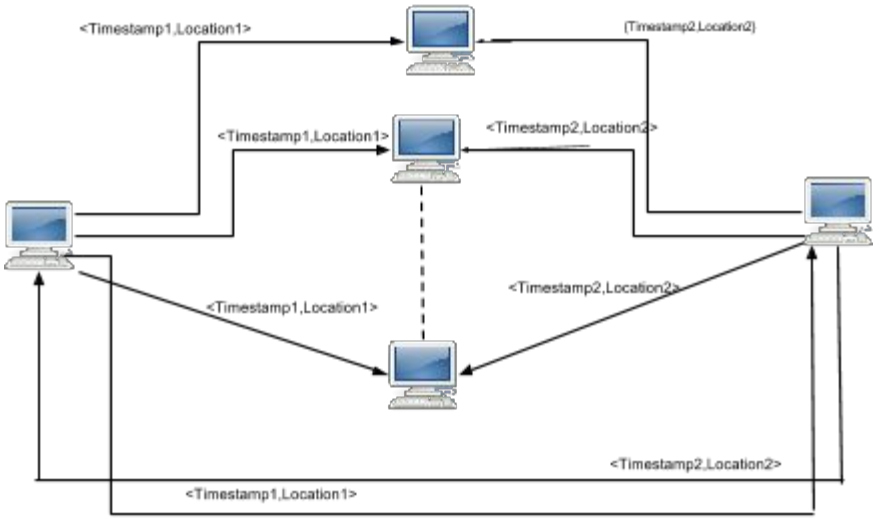
Algorithm Design:

Ricart and Agrawala's mutual exclusion algorithm:

- enter Robot at R_k
  - set StartState to *wanted*, where *wanted* is nearest neighbor location
  - multicast *"request"* with timestamp T_i and location L_i at R_k
  - wait for *"ok"* replies
  - change state to *Held* and enter the *Critical Section.*
- On receipt of a *request* $<T_i, R_j>$ at R_i where i $\neq j$
  - if(state = *Held*) or (state = *wanted* & (T_i , i) < (T_j , j))
  - add to *wait* queue
  - else

    send "*OK*" to R_i
- exit() at Robot R_i
- change state to "*Release"* and *"OK"* to all queued requests.





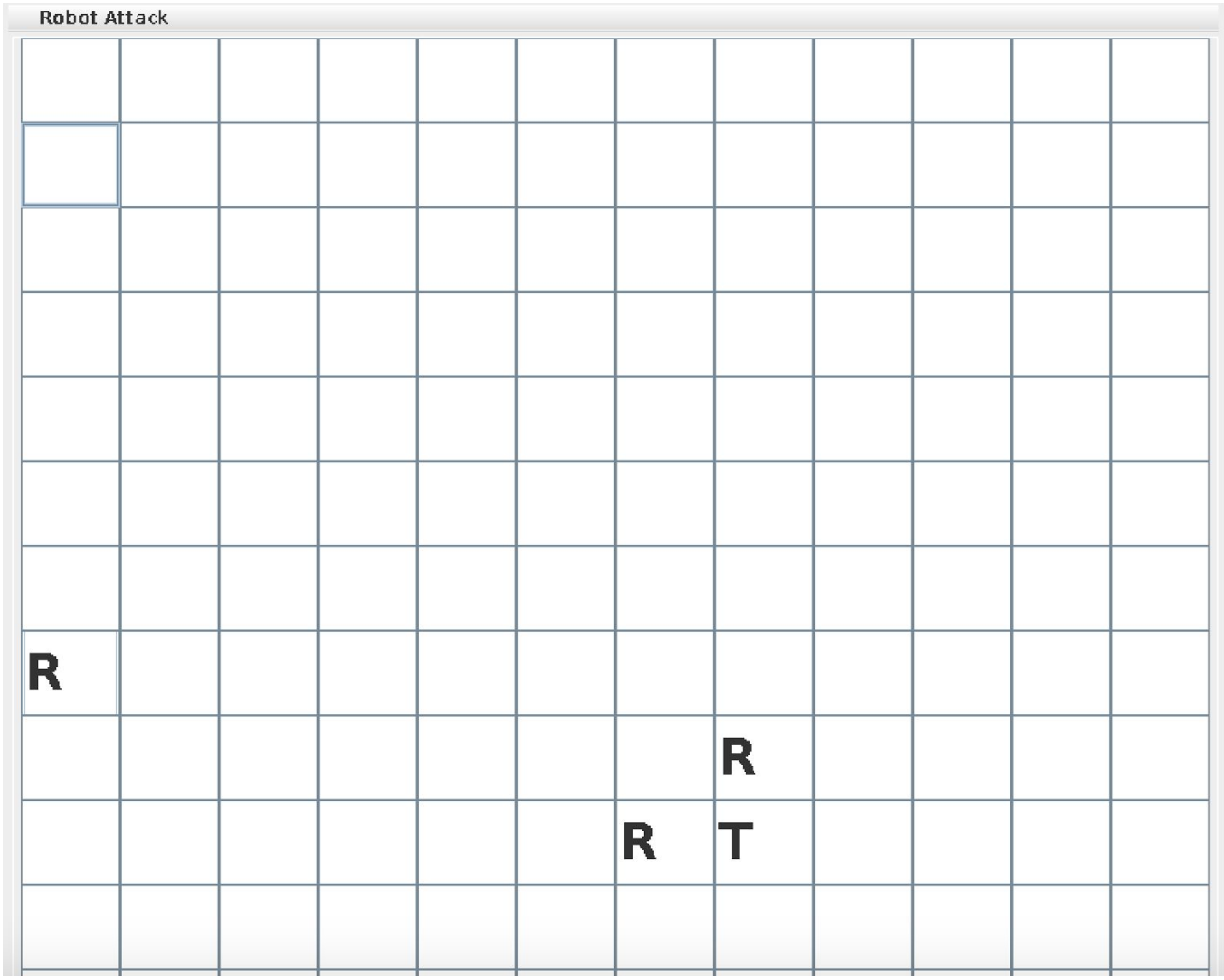
All the Robots will send their timestamp and location to each other Robots.

Byzantine Generals problem:

We first set whether a Robot is faulty or not at each Robot's console by taking an input from the command line and we will broadcast it to bootstrap server. After all the Robot's are connected to the server we call the byzantine generals functionality. Here, each Robot will send its target location to every other robot by multicasting the message as its target location. This message is received at every robot's listener and added into a list. We now calculated the most commonly occurring target location coordinates at each Robot's listener and send this value along with its target location to the bootstrap server. The server will now know that which robot is faulty if the common value it sends is not equal to its target location. If the robot is faulty then we just remove that robot's ip address from the robot list present at bootstrap and send back the updated list to all the other robots.

RESULT

Byzantine Output with malicious robot which is not moving:



DOCUMENTED CODE

```
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.net.InetAddress;
import java.net.Socket;
import java.util.Date;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Queue;
```

```
/**
 *
 * @author Sagar Shah
 * @author Gagandeep Malhotra
 * @author Sumedha Singh
 *
 */
public class RA {

	Robot robot;
	Location loc;
	DateNLocation ts;
	// Queue<String> mycsqueue= new LinkedList<String>();
	HashMap<String, Queue<String>> myqueues = new HashMap<String,
Queue<String>>();

	public RA() {

	}

	public RA(Robot robot) {
		this.robot = robot;
	}

	public void requestCS() {

		for (String destIp : robot.ipofrobot) {
			try {
				if
(!destIp.equals(InetAddress.getLocalHost().getHostAddress())) {
					Socket destSocket = new Socket(destIp,
Constants.ROBOT_LISTEN);
					ObjectOutputStream oos = new
ObjectOutputStream(destSocket.getOutputStream());
```

```
                                Date date = new Date();
                                robot.myRequestDate = date;

                                ts = new DateNLocation(date, robot.wanted);
                                oos.writeObject(new Message("Requesting CS", ts));
                                oos.flush();
                        }
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

        }
    }

    public void checkCS(String ip, DateNLocation dl) {
            if (robot.getmylocation().x == dl.loc.x && robot.getmylocation().y ==
dl.loc.y) {
                if (myqueues == null) {
                        myqueues = new HashMap<String, Queue<String>>();
                }
                if (!myqueues.containsKey(robot.wanted.x + " " + robot.wanted.y)) {
                        myqueues.put(robot.wanted.x + " " + robot.wanted.y, new
LinkedList<String>());

                }
                if (!myqueues.containsKey(robot.getmylocation().x + " " +
robot.getmylocation().y)) {
                        myqueues.put(robot.getmylocation().x + " " +
robot.getmylocation().y, new LinkedList<String>());

                }
                if (!myqueues.get(dl.loc.x + " " + dl.loc.y).contains(ip)) {
                        myqueues.get(dl.loc.x + " " + dl.loc.y).offer(ip);
                }
```

```
            } else {
                try {
                    Socket s = new Socket(ip, Constants.ROBOT_LISTEN);
                    ObjectOutputStream oos = new
ObjectOutputStream(s.getOutputStream());
                    oos.writeObject(new Message("Reply to CS", dl.loc));
                    oos.flush();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }

        }

        public void releaseCS() {
            if (myqueues.get(robot.loc.x + " " + robot.loc.y) == null) {
                return;
            }
            while (!myqueues.get(robot.loc.x + " " + robot.loc.y).isEmpty()) {
                try {
                    Socket socket = new Socket(myqueues.get(loc.x + " " +
loc.y).poll(), Constants.ROBOT_LISTEN);
                    ObjectOutputStream oos = new
ObjectOutputStream(socket.getOutputStream());
                    oos.writeObject(new Message("Reply to CS", "OK"));

                    oos.flush();
                } catch (IOException e) {
                    e.printStackTrace();
                }

            }
            myqueues.remove(loc.x + " " + loc.y);
```

```
        }

}
```

CONCLUSIONS

This project is a distributed robot attack simulation where the target is surrounded by n number of robots on board. With Ricart and Agrawala's mutual exclusion algorithm, the coordination between the robots is achieved in a distributed system . We have simulated the Robot attack in a distributed manner. We have successfully implemented following four cases :

1. Where the location of the target and each of the robots is known a priori.
2. Second where the location of the target is not known while the locations of the robots are known.
3. Third case where the location of the target is not known but the starting location of the robots is known.
4. There are one or more malicious robots and the malicious robot do not reach the target.

We have provided results in support of the implementation of our project.

REFERENCES

[1] Glenn Ricart and Ashok K. Agrawala. 1981. An optimal algorithm for mutual exclusion in computer networks. *Commun. ACM* 24, 1 (January 1981), 9-17. DOI=http://dx.doi.org/10.1145/358527.358537

[2] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.* 4, 3 (July 1982), 382-401. DOI=http://dx.doi.org/10.1145/357172.357176