DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING, SECOND CYCLE, 30 CREDITS

STOCKHOLM, SWEDEN 2017

Collision Avoidance for Virtual Crowds Using Reinforcement Learning

HALIT ANIL DÖNMEZ

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF COMPUTER SCIENCE AND COMMUNICATION

Collision Avoidance for Virtual Crowds Using Reinforcement Learning

HALIT ANIL DÖNMEZ

Master's programme in Computer Science
Date: July 1, 2017
Supervisor: Christopher Peters
Examiner: Hedvig Kjellström
Principal: Christopher Peters
Swedish title: Kollisionsundvikande för virtuella folkmassor som använder förstärkningslärande
School of Computer Science and Communication (CSC)

Abstract

Virtual crowd simulation is being used in a wide variety of applications such as video games, architectural designs and movies. It is important for creators to have a realistic crowd simulator that will be able to generate crowds that displays the behaviours needed. It is important to provide an easy to use tool for crowd generation which is fast and realistic. Reinforcement Learning was proposed for training an agent to display a certain behaviour. In this thesis, a Reinforcement Learning approach was implemented and the generated virtual crowds were evaluated. Q Learning method was selected as the Reinforcement Learning method. Two different versions of the Q Learning method was implemented. These different versions were evaluated with respect to state-of-the-art algorithms: Reciprocal Velocity Obstacles(RVO) and a copy-synthesis approach based on real-data. Evaluation of the crowds was done with a user study. Results from the user study showed that while Reinforcement Learning method is not perceived as real as the real crowds, it was perceived almost as realistic as the crowds generated with RVO. Another result was that, the perception of RVO changes with the changing environment. When only the paths were shown, RVO was perceived as being more natural than when the paths were shown in a setting in real world with pedestrians. It was concluded that using Q Learning for generating virtual crowds is a promising method and can be improved as a substitute for existing methods and in certain scenarios, Q Learning algorithm results with better collision avoidance and more realistic crowd simulation.

Sammanfattning

Virtuell folkmassimulering används i ett brett utbud av applikationer som videospel, arkitektoniska mönster och filmer. Det är viktigt för skaparna att ha en realistisk publik simulator som kommer att kunna generera publiken som behövs för att visa de beteenden som behövs. Det är viktigt att tillhandahålla ett lättanvänt verktyg för publikgenerering som är snabb och realistisk. Förstärkt lärande föreslogs för att utbilda en agent för att visa ett visst beteende. I denna avhandling implementerades en förstärkningslärande metod för att utvärdera virtuella folkmassor. Q Lärandemetod valdes som förstärkningslärningsmetod. Två olika versioner av Q-inlärningsmetoden genomfördes. Dessa olika versioner utvärderades med avseende på toppmoderna algoritmer: Gensamma hastighetshinder och ett kopieringssyntestillvägagångssätt baserat på realtid. Utvärderingen av publiken gjordes med en användarstudie. Resultaten från användarstudien visade att medan Reinforcement Learning-metoden inte uppfattas som verklig som den verkliga publiken, uppfattades det nästan lika realistiskt som massorna genererade med Reciprocal Velocity Objects. Ett annat resultat var att uppfattningen av RVO förändras med den föränderliga miljön. När bara stigarna visades upplevdes det mer naturligt än när det visades i en miljö i riktiga värld med fotgängare. Det drogs slutsatsen att att använda Q Learning för att generera folkmassor är en lovande metod och kan förbättras som ett ersättare för befintliga metoder och i vissa scenarier resulterar Q Learning algoritm med bättre kollisionsundvikande och mer realistisk publik simulering.

Contents

Chapter 1

Introduction

This chapter introduces the problem that was investigated, describes the limitations and how the research question was answered.

1.1 Background

Virtual crowds are used in many applications ranging from video games to movies. The purpose of these crowds are either to add an atmosphere to the environment or to display a certain behaviour according to the user's commands. These behaviours are defined by the developers or according to a predefined scenario. In architectural designs, some pedestrians are added to the model in order to show how the constructed area will look like. In video games, crowds are either armies the user can control or they are alive crowds where several random events happen and in movies, they are used for creating massive numbers of soldiers marching or fans filling a stadium (see images below for sample usages). There are software available for generating virtual crowds. Two of them are called "Golaem" and "Massive Software".

Generated virtual crowds needs to behave realistically since virtual crowds aim to look similar to a crowd in real world. Realism depends on the environment and the scenario the crowd is generated for. In a stadium for example, a realistic example would be to go towards the nearest exit knowing that there will be more agents. In a game however, realistic behaviour could be to avoid such situations. Achieving realism is a challenge for generating virtual crowds. Another major challenge is to avoid collisions. This condition applies to each scenario

Figure 1.1: From left to right, Screenshot from Golaem software [11], A Battle scene with virtual agents using Massive Software [38] and City populated by crowd [10]

a crowd is in since it is impossible for humans, for example, to go through each other. Collision avoidance therefore is important for virtual crowds. Achieving collision avoidance however does not necessarily mean achieving realism. Depending on the algorithm, an agent can avoid colliding with another agent yet it can do so in a manner where no human would behave to avoid collisions. This problem creates another challenge which is to have a collision avoidance algorithm that also results with realistic movements and behaviours.

The question of realism or naturalness of the crowds can be evaluated by making a perception study. In this study, several users watches a video of the generated crowd and fills in a questionnaire about them. These questions are about the way pedestrians behaved in the crowd, their behaviour while navigating and the general overview of the crowd i.e how natural it looked like. Results from these studies can be analysed scientifically and therefore can be used to evaluate the generated virtual crowds.

There are different approaches to the virtual crowd simulation. These methods range from particle based models to fluid based models. RVO library for virtual crowd simulation is one of the most commonly used library.

Reinforcement Learning was proposed as an alternative to the existing crowd simulation methods. Motivation behind this approach was being able to use the experience of one agent thus reducing the workload for creating a crowd simulation. Also, Reinforcement Learning can be used in any environment since it is represented as states agent can be in. However, Reinforcement Learning method was not evaluated and compared with the existing methods. This proposed method was not evaluated.

1.2 Thesis Objective

Objective of this thesis therefore is to answer the research question: "can Q Learning create plausible crowds comparing to Reciprocal Velocity Objects(RVO) and real crowds". In order to reach this goal, virtual crowds using three different methods were generated. First method was Q Learning where agents in the virtual crowd used Q Learning for navigation and collision avoidance (see section 3.1). Second method was RVO where RVO library was used for virtual crowd simulation (see section 3.2). Third method was virtual crowds based on data from the real world crowds[24].

This thesis will investigate the ease of use for the Q Learning method as well. Ease of use in this context refers to how many parameters needed to be set in order to generate virtual crowds using Q Learning and RVO.

For this thesis, naturalness means: How close the virtual crowd to a crowd in the real world. Meaning, if a crowd is natural, that crowd can be observed in the real world. If it is not natural, then there is no crowd in real world that will behave the way generated virtual crowd behaves.

Evaluation of the virtual crowds will be done with a user study where a low density crowd will be generated and users will be asked to answer several questions about their perception.

Virtual crowd generated with Q Learning method should be able to avoid collisions with other agents in the scene and should be able to reach to the goal location.

1.3 Delimitations

There are many approaches to the virtual crowd generation. Reinforcement Learning, RVO and a copy-synthesis approach based on real-data is taken into consideration for this thesis. RVO is already implemented as a library and the implementation was taken from the online source of the method[1]. A project was done using a copy-synthesis approach based on real-data from the crowds to generate new crowds [24]. The outcome of this method was provided as a video to be used in the user study.

[1] http://gamma.cs.unc.edu/RVO2/

Casadiego et al. proposed using Q Learning for generating virtual crowds[7]. This thesis is using the proposed states, actions and reward calculations by the mentioned work. However, work of Casadiego et al. lacks the evaluation of the virtual crowd. Since Q Learning is not dependent on a specific environment, results can be easily reproduced and evaluated. Only modifications will be done with the training of the agent and calculating rewards for a state the agent is in.

Results of this thesis is based on the assumption that the proposed method for the Reinforcement Learning is accurate and RVO library is working properly according to the proposed algorithm.

Virtual crowds are low density crowds i.e there are 2 agents for one scenario and 10 for another scenario. These two scenarios are explained in detail at the evaluation section.

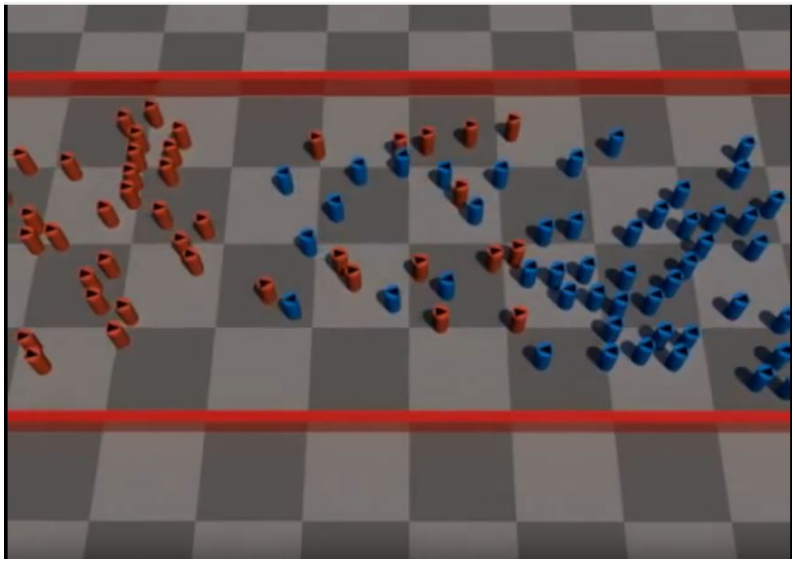

Figure 1.2: A corridor scene. Cylinder in red is going towards right and ones in blue is going towards left. Taken from [18].

1.4 Choice of Methodology

Evaluation of the virtual crowds generated in this thesis will be done with a user study. Several crowds will be generated and resulting crowd will be recorded as a video to later display to the users for evaluating them.

Q Learning requires states, actions and rewards for these actions [42]. How an action for the current state is rewarded depends on how rewarding function is defined. This reward function can be changed and this enables Q Learning to generate different types of virtual crowds using different reward approaches. Environment is represented in the form of states in Q Learning. Therefore, there is no need to create a specific environment for using Q Learning and it is enough for an agent to be able to perceive the state it is in. Another advantage of the Q Learning is being able to store old values and re-use them. Agents can load these values and use them for navigation without needing to be trained again. This feature is useful for conducting a user study as well where old values can be used for creating different crowd simulations for comparison.

RVO can be used to create collision avoiding crowds with behaviours similar to real crowds [5]. This library is available for the same programming language supported by Unity Game Engine[2] which made it possible for it to be implemented alongside with Q Learning, enabling the Q Learning to be compared with another method.

This thesis is organized in the following way. First, relevant works about crowd simulation will be presented including methods for evaluating virtual crowds. Then implementation details will be presented. The implementation chapter will be followed by the evaluation chapter. Evaluation chapter will contain the results gathered from the user study and analysis of these results. Finally, conclusion chapter will be presented where results will be discussed and several ways the thesis can be carried further will be presented.

[2]https://unity3d.com/

Chapter 2

Related Work

This chapter gives information about the previous work on the field of this thesis. Then describes the theory of the used algorithms in detail.

Since Reinforcement Learning, especially Q Learning, RVO and data driven crowds were used in this thesis, sections for these methods contains more details.

2.1 Social Forces

Social forces, proposed by Helbing et al. are "forces" that are not applied from the environment[15]. They are the actions that were performed by the individual pedestrians. All these forces are modelled as terms of formulations [15]. Below, these forces are given that applies to the pedestrian α.

First force is the force applied towards the goal. Pedestrian α will want to reach to its destination with the shortest possible route. Therefore, the route taken has to be the one without detours. The goal of the pedestrians can be thought of gates or areas rather than points.

Second force is the force of the other pedestrians. This force is due to the density of the crowds and the personal space of the pedestrian which is interpreted as territorial effect. This is the force which is being applied when a pedestrian gets too close to another pedestrian who can be a stranger.

Another environment factor to consider are the walls. Pedestrians should avoid walls or borders of the buildings (streets, obstacles etc.). These factors will invoke a repulsive factor in the pedestrian.

Pedestrians can be attracted to each other as well. This can occur when a pedestrian saw someone who he/she is familiar with (family, friends, artists etc) or by objects.

Remaining factor to consider is the location of these forces. Events happening behind the pedestrian will not affect with the same impact.

These forces are calculated for the individual pedestrians to form a crowd behaviour. Crowds using these forces results with realistic behaviours [15].

2.2 Steering Approaches

Steering approaches determines how virtual agents navigates through the virtual world. Below are different steering approaches are given.

Reynolds proposed several steering approaches for the virtual characters[34]. These proposed steering approaches are defined according to the steering approaches of vehicles. Along with vector flows and collision avoidance behaviours (see figure 2.1).

Path following, collision avoidance, fleeing, separation flow field following are among the proposed steering algorithms. These steering behaviours were proposed with the purpose of animating virtual agents. In order to move, the agent must first get the position of the destination, then according to the environment choose its path and move there according to the movement type defined.

Steering approaches comes into play when the agent needs to calculate a path towards the goal while avoiding collisions.

Another approach to the steering of the crowds is using the visual perception of the agent [29]. This approach takes into account the current perception of the agent in the world and steers away from them. The idea is to make the agents behave like in the real world where the only thing the pedestrians are aware

of is what they perceive within their radius of vision (see figure 2.1).

It was proposed that the crowds behave like fluids in certain scenarios [28]. In order to improve the simulation of a virtual crowd, each agent was treated like a particle in a fluid. Then all agents within the simulation behaves like a fluid with a high density (see figure 2.1).

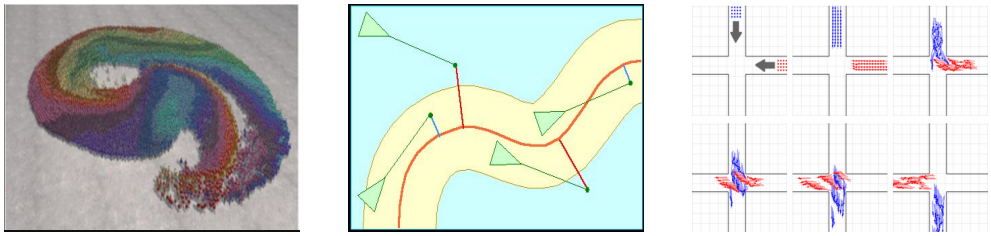

Figure 2.1: (left)Aggregate simulations crowds. From [28] (center)Steering behaviour of Agents. From [33] (right)City populated by crowd. From [29]

2.3 Continuum Crowds

This method is proposed by Treuille et al. Idea is to calculate a dynamic potential field in the environment that takes the obstacles and all the agents into account to generate a path for the virtual agents [39].

This simulation calculated the paths of the virtual agents based on several assumptions. They are given below.

Assumptions made for the individual agents

Goal of an agent is a geographical location expressed with proper coordinates. Goal is not vague like saying "find an empty seat in a movie".

Virtual agents move with the maximum speed

There is a *discomfort field* that makes the virtual agents want to take an alternative path. For example, when crossing the road, most pedestrians prefer the way with a pedestrian crossing. This holds for the areas in front of the agents as well since they should not walk in front of each other.

When choosing a path, the length of the path, amount of time it takes to travel from the selected path and the discomfort felt per unit time is taken into account.

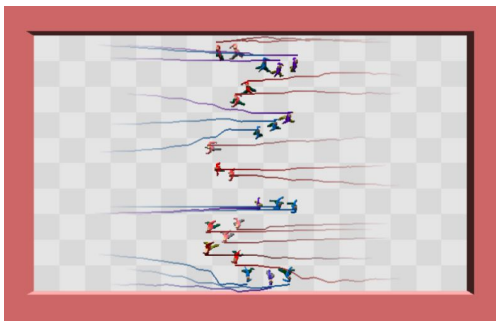

Figure 2.2: Two groups passing by each other. From [39].

Finding the optimal path In order to find the optimal path a potential for the agent was calculated starting from the goal while integrating the cost of the path along the way. For moving the agents in the crowd, the crowd was divided into set of groups and at each time step, a potential field was constructed for each group. After construction, agents in the groups moved accordingly.

Speed of the Agents Speed here is density dependent. At low densities it remains constant on flat surfaces and changes with the slope. At high densities, movement is inhibited when trying to move against the flow and it is not affected when moving with the flow.

Results Continuum Crowds do not properly simulate the crowds with large densities since, the force between the pedestrians dominate the physical forces. Yet, for the smaller densities, they have proved to be quite effective[39]. A test case was the simulation of the traffic lights where agents, as expected, did not choose to take any other route once the traffic lights turned red. Simulation was tested with evacuation and school exits where it provided good results as well[39]. Figure 2.2 is the output of the continuum crowds.

2.4 Cellular Automata Models

This is an artificial intelligence method for simulating multi agent systems. Agents are placed in a space which is represented as uniform cells (see figure 2.3). Several rules are defined for the behaviour of the agents and they define the state of the agent[8]. Agents can only perceive the environment around them and they are never aware of each other. This method gives accurate results for low or medium density crowds yet for high density crowds it fails on several cases.

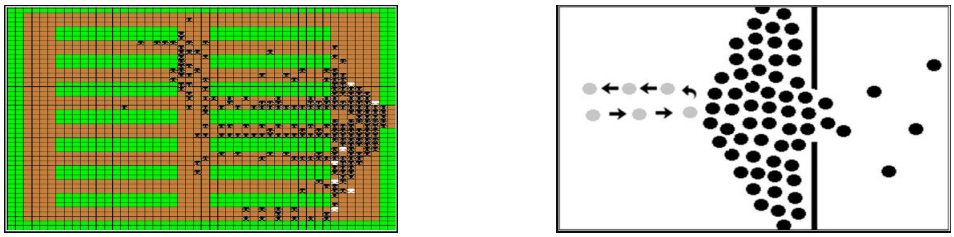

Figure 2.3: (left)An emergency simulation with Cell automata using a grid. From [45] (right)Agent flocking to a door. From [32]

Cellular automata approach was implemented for several crowd simulation and evacuation scenarios in [6] [45] [32]. See figure 2.3 for this approach.

2.5 Particle system based crowds

The main idea with the particle system based crowds is that the crowd as a whole is modelled using the behaviour of a particle system [13] (see figure 2.4 for an output of the model). An example of this system was proposed by Helbing et al [14]. Their proposed model is called the Helbing model.

Helbing model was tested for emergency or panic scenarios and concentrated on simulating several occurrences within these cases. In the tested model, generated virtual crowds behaved similarly with the real crowds[14].

Other similar particle system based model was implemented by Heïgeas, Laure, et al. for the output of said model. With this model, it is possible to explain the interactions between the pedestrians as physically based forces exerted between each pair of pedestrians or

particles. This can be described as; for large number of pedestrians, behaviour will be to spread out and decrease the density. In narrow places, pedestrians will tend to come closer to each other.

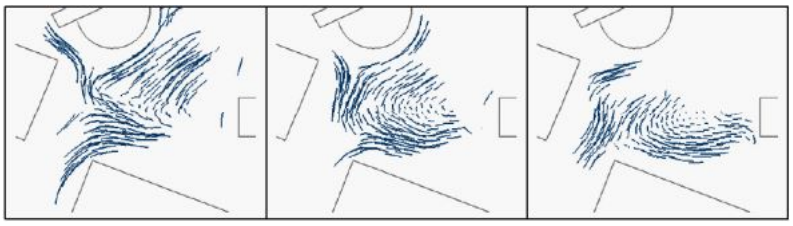

Figure 2.4: Crowds simulated in an emergency situation with particle based models. From [13]

Obstacles in the simulation were represented as sets of fixed particles that interacts with pedestrians. The perception of the obstacle is also calculated. For example, if the building is in front of the pedestrian and he/she sees it, applied effect will get larger.

2.6 Bayesian Decision Process

Another way of generating virtual crowds was to use Bayesian Decision process [25] proposed by Metoyer et al. This method was proposed for the purpose of populating model designs for building environments and buildings.

Environment Agents that were generated to the environment navigated in a motion field. This motion field was defined by the user. The fields also provides a basic level of behaviour for the agent. Motion fields acts like social forces for the pedestrians. The social force model of [14] was used for this purpose.

Each agent in the simulation was defined as a point mass dynamic.

Navigation In order to define a state the pedestrian is in, seven discrete features were defined. They are:

1. Is the path around left blocked by other pedestrians or obstacles ? (Y or N)
2. Is the path around right blocked by other pedestrians or obstacles ? (Y or N)

3. Relative speed of the colliding pedestrian
4. Approach direction of the colliding pedestrian
5. Colliding pedestrians distance to collision
6. Pedestrians distance to collision
7. Desired travel direction

Based on these seven features, a naive Bayes classifier was defined.

Decision trees along with naive Bayes was also tested. Node of the tree was one of the seven features described above. Children of a node is each possible value of a feature. Leaf nodes represents the decisions. Tree was built by a top down greedy search that chooses the best feature to test at the root and each subsequent level.

Results Examples to test the approaches were provided by a user. In total there were 146 examples over several scenes. Success for test was defined as the agent picking an action that was provided by the user saying how the agent should behave. Naive Bayes decision-making gave 76% accuracy while the decision tree algorithm gave 92% accuracy.

2.7 Reciprocal Velocity Obstacles

Reciprocal Velocity Obstacles (RVO) is a concept for multi-agent navigation [4] which is an extension of velocity obstacle concept [9]. RVO extends this by taking the reactive behaviour of the other agents by assuming that they will also use a similar collision avoiding reasoning. RVO method was used in this thesis for virtual crowd generation. Therefore, this section will describe RVO in more detail.

Velocity Obstacles

Assume there is an agent A with a reference point P_a. Let B, be a moving obstacle with reference point at P_b. The velocity obstacle of B to agent A is then the set consisting of all the velocities for A that will result in a collision with B at some time T with B's velocity V_b.
Geometrically, a velocity object can be defined as the Minkowski

sum of two objects A and B. Let λ(p,v) denote a ray starting at p and heading in the direction of v: λ(p,v) = {p+ tv | t $\geq$ 0 }. If this ray is intersecting with the Minkowski sum of B and -A centered at p_b, then the velocity v_A is in the velocity obstacle of B.

Then we can define the velocity obstacle of B to A as: VO $\overset{A}{B}(v_b)$ = { v_a | $\lambda(p_a, v_a\text{-}v_b) \cup$ B $\oplus$ -A $\neq \emptyset$}

This equation tells that A and B will collide at some point. If v_a is outside the velocity obstacle, they will not collide. If v_a is at the boundary, it will touch B at some moment in time. It can be used for navigation as following: Agent chooses a v_a that falls outside the velocity obstacle of B, among those values, it chooses the one that is most directed towards the goal.

Reciprocal Velocity Obstacles Properties

1. **Definition** RVO chooses a new velocity for an agent that lies outside the other agent's velocity obstacle. This definition can be formalized as the following:

$$RVO_A^B(\mathbf{v}_B, \mathbf{v}_A) = \left\{ \mathbf{v}'_B | 2\mathbf{v}'_B - \mathbf{v}_B \epsilon VO_A^B(\mathbf{v}_B) \right\} \tag{2.1}$$

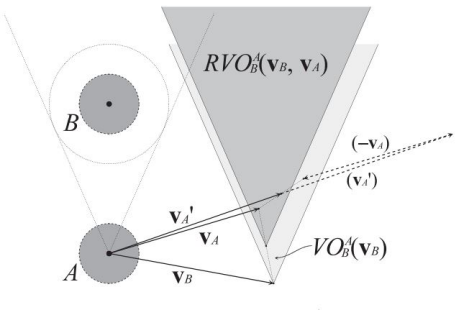

Figure 2.5: $RVO_A^B(\mathbf{v}_B, \mathbf{v}_A)$ of agent B to agent A .

The reciprocal velocity obstacle $RVO_A^B(\mathbf{v}_B, \mathbf{v}_A)$ of agent B to agent A has all the velocities for A that are average velocities and velocities that are outside agent B (see figure 2.5). It can also be interpreted as the following: $\frac{\mathbf{v}_A + \mathbf{v}_B}{2}$.

2. **Generalized Reciprocal Velocity Obstacles** In some cases, some agents may have priority over some other agents. RVO

can be generalized to cover changing priorities as well. Assume the effort by agent A denoted by α_B^A. Agent A needs to choose the *weighted* average of $1 - \alpha_B^A$ of its current velocity $\mathbf{v}_A$ and α_B^A of a velocity outside the velocity obstacle $VO_B^A(\mathbf{v}_B)$ of agent B, and agent B chooses a velocity that is the weighted average of $1 - \alpha_A^B = \alpha_B^A$ of its current velocity $\mathbf{v}_B$ and $\alpha_A^B = 1 - \alpha_B^A$ of velocity outside the velocity obstacle $VO_A^B(\mathbf{v}_A)$ of agent A.

Based on this information, generalized Reciprocal Velocity Obstacle of agent B of agent A is defined as follows:

$$RVO_B^A(\mathbf{v}_B, \mathbf{v}_A, \alpha_B^A) = \left\{ \mathbf{v}'_A | \frac{1}{\alpha_B^A}\mathbf{v}'_A + (1 - \frac{1}{\alpha_B^A})\mathbf{v}_A \epsilon VO_B^A(\mathbf{v}_B) \right\} \tag{2.2}$$

Multi-Agent Navigation

The general approach to navigation is first to choose a small time Δt within the time step of the simulation. For each agent in the simulation that has not reached its goal, a new velocity is selected independently during the simulation and the positions of the agents were updated accordingly.

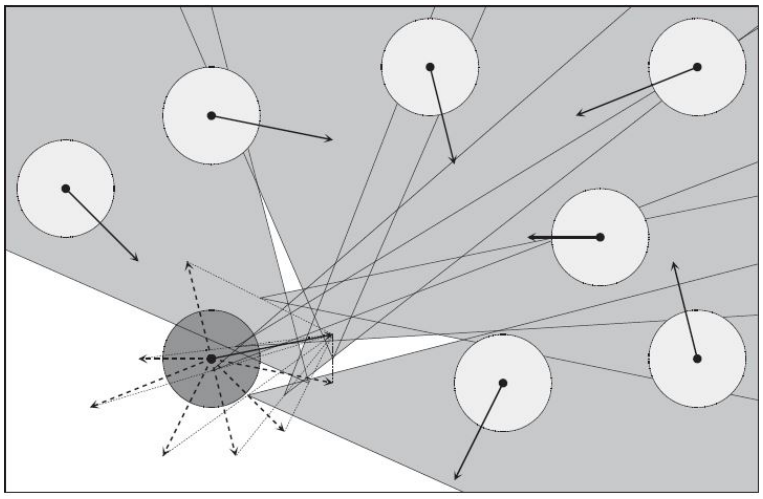

Figure 2.6: The combined reciprocal velocity obstacle for the agent (dark) is the union of the individual reciprocal velocity obstacles of the other agents. From [4]

Combined Reciprocal Velocity Obstacles

So far, Reciprocal Velocity Obstacle was defined for pairs of agents. Combined reciprocal velocity obstacle RVO^i for agent A_i becomes the union of all reciprocal velocity obstacles calculated for all the other agents individually and the velocity obstacles generated by moving or static obstacles.

In order to avoid collision, agents have to select a velocity outside its combined reciprocal velocity obstacle. However, agent properties may be subject to several restrictions that will limit the available velocities. If an agent is subject to a maximum speed v_i^{max} and a maximum acceleration a_i^{max} the set of admissible velocities are defined by

$$AV^i(\mathbf{v}_i) = \{\mathbf{v}_i' | \|\mathbf{v}_i'\| < v_i^{max} \wedge \|\mathbf{v}_i' - \mathbf{v}_i\| < a_i^{max}\Delta t\} \qquad (2.3)$$

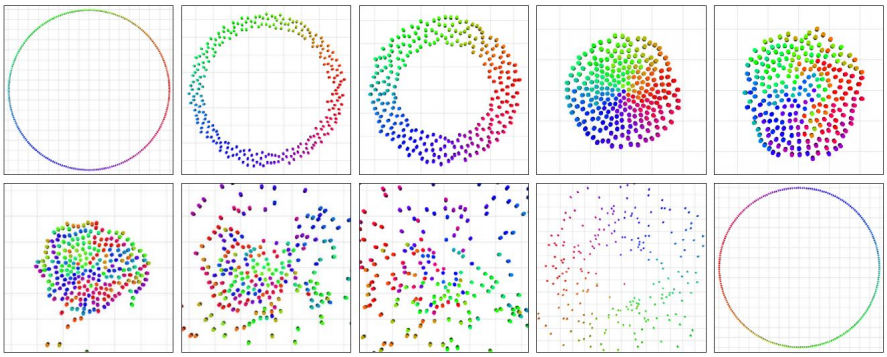

Figure 2.7: Output of the RVO with a circle scenario with 250 agents. Each agent's goal is at the opposite end. From [4]

Optimal Reciprocal Collision Avoidance

Optimal Reciprocal Collision Avoidance (ORCA)[30], proposed by [40], is a formal approach to reciprocal n-body collision avoidance. The algorithm was tested both on robots and virtual agents. Both RVO and ORCA are velocity based collision avoidance methods. ORCA enables several agents to avoid collision with each other unlike RVO.

Working idea of ORCA for an agent is as follows: Agent gets the radius, current position and the current optimization velocity of

the other agents. Then it calculates the velocities that avoids collisions and moves towards the goal and selects the best possible velocity among the calculated velocities.

Since the computations were done individually for each agent, it was possible to parallelize the processes and separate them among the processors. This made the algorithm faster. Agents in the virtual world was able to avoid collisions with each other smoothly. This algorithm was tested for robots in which the results were the same with the virtual agents and robots were able to change velocities smoothly to avoid collisions[40].

2.8 Reinforcement Learning

Reinforcement Learning was used for training an agent to play and complete a game[27]. The work of Mnih et al. is the first usage of the Reinforcement learning to play an Atari game which includes navigation of the agent while avoiding obstacles. The environment is perceived by processing the pixels in the game. Learning was done with Deep Q Learning and not the Q Learning itself.

Google used Reinforcement Learning to create an agent that plays board games [35] [43].

The Q Learning method was first proposed by Watkins in 1989[43] and the proof of convergence was done by Dayan et al. [42] in 1992. Another usage was to train an agent to imitate a locomotive action to navigate in terrains[31].

Huang et al. proposed a system using Q Learning to make a robot navigate while avoiding the obstacles in its way [16] where resulting values from Q Learning were stored in a network and not in a table contrary to Q Learning. Smart et al. trained a robot to reach to a goal in an obstacle free environment and in an environment with one obstacle[37]. Beom et al also used Reinforcement Learning to make a robot navigate in an unknown environment[2].

2.8.1 Q Learning

In Q Learning, environment is seen by the agent as a set of states and there is no dependence on the environment itself[43]. An agent has actions that can be taken and each action taken for a state results with

a reward. A Q value is then can calculated and written to the action for the state. These Q values are stored in a table which is called a Q Table.

Using Q Learning to simulate virtual crowds was proposed by Casadiego et al. and the approach that was introduced will be the basis of this thesis in order to evaluate the proposed method. In the following sections, the method used in [7] will be described.

States

States are separated into two. First one is obstacle state and the second one is the goal state. State the agent is in is calculated according to the following equation.

$$agentState = \begin{cases} (numDistances + 1)^{nrOs} * numGoalState & \text{not at final state} \\ (numDistances + 1)^{nrOs} * goalState + obstacleState & \text{at final state} \end{cases}$$

$nrOs$ is number of obstacle state which is 8. Final state count for this approach was calculated with the following equation in [7]:

$$nrGoalStates * ((nrdistances + 1)nrOS)) + 1 \tag{2.4}$$

With 8 digits in occupancy code, 2 distance states and 7 goal states, total state number was 52,489 states according to the equation above. For perceiving the obstacles, front of the agent was considered more important and was represented with more intervals. 2 distance states were chosen in order to differentiate between obstacles that are close and far away. Goal states are chosen as 8 since they can represent the position of the goal relative to the agent's position.

Obstacle State Obstacle state is calculated by converting an 8 digit number, which is called an occupancy code, into a single number. Figure 2.9 is the occupancy code of the figure 2.8.

Agent has two different radii. Outer radius which is at the largest distance from the center of the agent and the second one which is coloured with blue is called the inner radius. Objects in red are obstacles.

If there is an obstacle that is intersecting with the inner radius of the agent, 2 is placed in the corresponding place in the occupancy code. If an obstacle is intersecting with the outer radius only, then 1 is placed

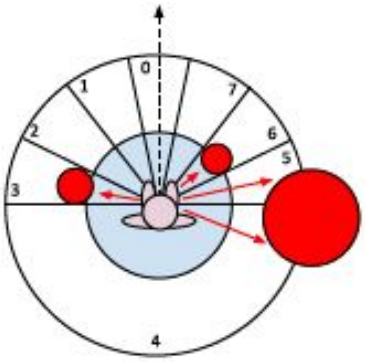

Figure 2.8: An agent in the environment with static obstacles. Image from [7]

Angle Interval	0	1	2	3	4	5	6	7
Code	0	0	0	2	1	1	2	0

Figure 2.9: Resulting occupancy code. Image from [7]

in that interval. Converting the occupancy code to a single number is done with the following equation.

$$obstacleState = \sum_{i=0}^{nrOS-1} OccupancyCode[i] * ((nrdistances + 1)^{nrOS-1-i}) \tag{2.5}$$

$nrOS$ is the number of intervals in the occupancy code which is 8 and $nrdistances$ is the number of distances which is 2 (1 and 2 for filling the occupancy code).

Goal State Goal state is calculated by taking the angle between the vector that is pointing towards the goal position and the vector that is facing forward from the current position of the agent. This angle is then placed within intervals. There are 8 possible actions the agent can take. Which ever interval the degree falls, that number is returned (see figure 2.10). A number between 0 and 8 is returned for this state.

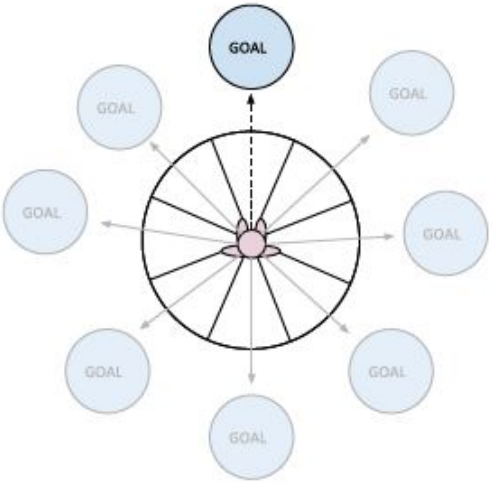

Figure 2.10: Goal state calculation. Image from [7]

Actions

Actions agent can take is defined in the figure 2.11. Selecting an action is done with a policy. An example of a policy is ϵ - greedy policy. With this policy, the action that has the highest Q value is selected at the state agent is in. Since this policy was used in the work of Casadiego et al. it was also used in this thesis.

Reward

Rewards are calculated with the general formula: $rewardObstacle + rewardGoal$.

Goal reward Goal reward is calculated by taking the angle between the forward facing vector from agent's current position and the vector facing towards the goal position from the agent's position.

$$rewardGoal = cos(angleBetween(\overrightarrow{currentdirectiontogoal}, \overrightarrow{currentdirection}))^3 \tag{2.6}$$

Obstacle Reward Obstacle reward is calculated with the following formula:

$$obstacleReward = -\sum_{i=0}^{nrOS-1} 10/10^{(DOC_i-1)*2} \tag{2.7}$$

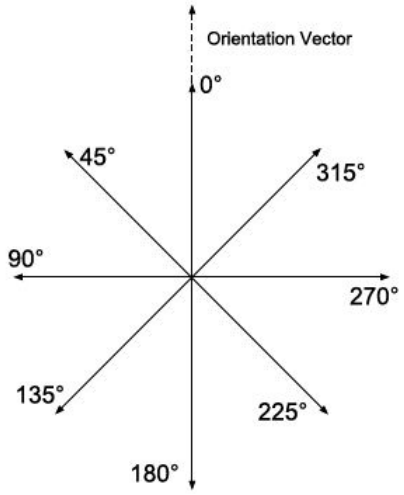

Figure 2.11: Possible actions agent can take

Results Result of this approach resulted with pedestrians avoiding collisions and reaching to their goal. Generated virtual crowds by Casadiego et al. is not evaluated however.

2.8.2 Deep Q Learning

Deep Q Learning is a variant of Q Learning. It was used first for playing an Atari game [27] [26]. The game environment was perceived as pixels. A neural network (a computer system modelled on the human brain and nervous system) is created for representing the Q values for a state. In this approach, a state and the action is given to the network and the network returns the Q value. There is no table structure in this method. It is therefore possible to represent more states for the agents.

When a new state is detected, this method makes it possible to look back at the gained experience. It is therefore possible to make better decisions for a state.

2.9 Universal Power Law

Universal Power Law is another method for generating crowds[19]. Unlike Q-Learning and RVO, this method uses the behaviour of the

particle systems and then simulates the virtual agents accordingly. Each virtual agent has an interaction energy between other virtual agents.

The occurring energy and interaction is based on the projected time to a future collision. Based on this analysis, Universal Power Law is able to simulate virtual crowds with several simulations[19]. See figure 2.12 for such an example.

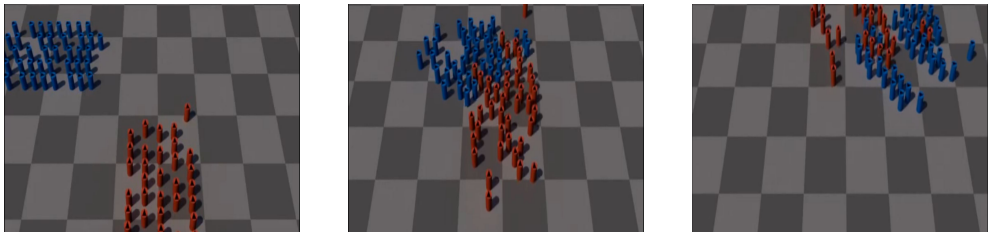

Figure 2.12: Universal Power Law algorithm simulating collective motion. Taken from [17]

Data Analysis

In order to analyse the interaction energy between the pedestrians, Karamouzas et. at [19] used the datasets gathered from real crowds. First, noise in the data was removed. This was done by gathering data sets with similar densities resulting with 1146 trajectories of pedestrians in sparse to moderate outdoor settings and one Bottleneck data set with 354 trajectories of pedestrians in dense crowds passing through narrow bottlenecks.

After the datasets were gathered, they were analysed with statistical-mechanical pair distribution function. If this function was smaller than 1, it indicated collision avoidance.

Pair distribution function of the Cartesian distance between agents provided an appropriate description of the interaction between pedestrians. If this function had a small value, it meant that agents had a strong interaction.

Generated Crowds

Universal Power Law provided good results with crowds that has low to medium densities[19]. Yet for the crowds with very high densities, this algorithm failed to provide a proper simulation for some scenarios. Investigation for this was left for future work by the authors [19].

2.10 Data Driven Crowd Simulation

Data driven crowd simulations are simulations generated by extracting the trajectories or other relevant data from an observation of a real crowd and generate agents that acts accordingly to the behaviour in the extracted data.

1. An approach to this is to use recordings of the crowds and extracting the trajectories [44] [21] [3]. After extracting the trajectories, agents in the virtual world moved according to these trajectories.

   Lee et al. extracts the trajectories in a semi-automatic way. A tracking algorithm was implemented. In order to use the algorithm, start and end position of the pedestrian needed to be given to the system.

   Decision making of the agents were then made in a form of finite state machine. State machine representation requires the agents to adapt to their environments. For this purpose a framework was developed which allowed the user to give generated agents and the surrounding objects a behaviour type. Agents then choose the trajectory models accordingly.

   Generated crowds gave promising results. They were tested with evacuation, small town, ant behaviour and previously simulated crowds. For each of these cases, algorithm produced realistic results.

2. Same approach with the data was implemented as an example based crowd [22]. Trajectories were gathered from the footage of the real crowds. Then according to the trajectory, an agent was able to make decisions accordingly.

   Difference of this method is the fact that it solely computes the trajectory based on the trajectory of the agent that was currently at the agents position. A query is made to the example datasets that was generated from the extracted crowd data. Based on how similar a query is to a given agent, that trajectory was assigned to the agent at the position the agent sent.

   Generated paths were checked for collisions. If a collision was possible, then the agent chose a trajectory that will prevent a collision.

Resulting crowd simulations were able to avoid colliding with each other. Also, it was able to simulate behaviours like shopping or roaming freely. It was also able to simulate given scenarios with no collisions or any other unnatural phenomena.

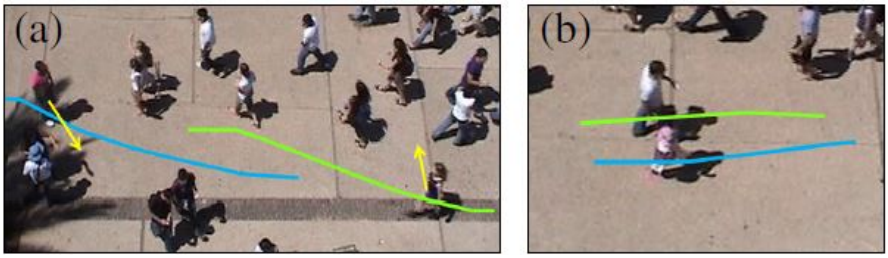

Figure 2.13: Image from a data set of real crowds. From [20]

There are several recorded data from the real crowds available.

2.11 Virtual Crowd Evaluation Methods

2.11.1 Entropy Metric

Entropy metric is a statistical evaluation method and it is proposed by Guy et al. [12]. Evaluation of the virtual crowds are done according to their positions in the simulation. This also involves comparing the calculated pedestrian position from the given simulator and with the current position of the pedestrian in the given data which holds the correct positions of the pedestrians.

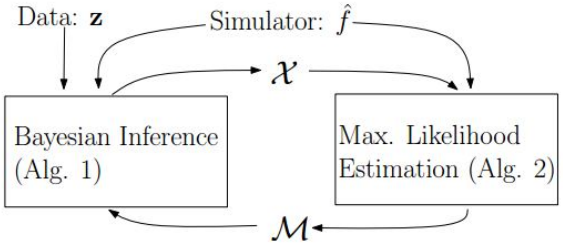

Figure 2.14: Algorithm for Entropy Metric. From [12]

This evaluation method works in two steps. First step is to estimate a distribution of the simulation states to represent the given data. Second step is to use the given crowd simulator to predict the subsequent state from the preceding state (see figure 2.14).

Paths and locations in a crowd are constantly changing therefore these were guessed. Based on the observations from the crowds at a time, next step is generated using the simulator and this gives an approximation of the actual crowd data. Noise was removed from the real data. Using the current state of the crowd, next location in the future is calculated using the crowd simulator with an error rate. These errors create an error distribution within the crowd data set.

Assuming we have $\mathbf{z}_k$ as the observed state in the crowd, these points are converted to true crowd state $\mathbf{x}_k$(deprived from noise). Next state of the crowd will be

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k) = \hat{f}(\mathbf{x}_k) + \mathbf{m}_k, \mathbf{m}_k \sim \mathcal{M} \tag{2.8}$$

where $\mathbf{m}_k$ is the error rate, $\hat{f}$ is the crowd simulator that user defines, $\mathbf{x}_{k+1}$ is the next crowd state, f is the actual crowd function and $\mathcal{M}$ is the error distribution. If the error rate is low, then the crowd simulator is better.

Entropy Metric is then defined as the entropy of the distribution $\mathcal{M}$ of errors between the evolution of a crowd predicted by a simulator $\hat{f}$ and by the function f (see figures 2.15 and 2.15).

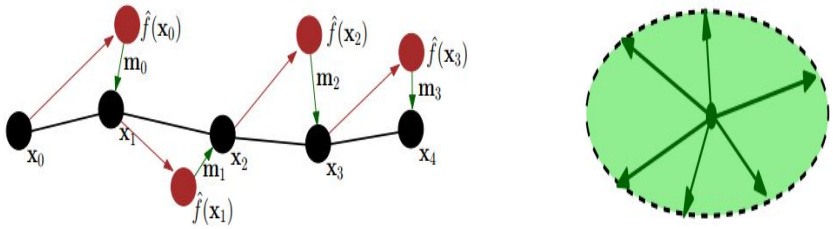

Figure 2.15: (left)Prediction error in each time step. Taken from [12] (right)Distribution of error over time steps. Taken from [12]

Results Guy et al defined several criteria for analysing the results gathered. They are given below.

Rankable Results: All the simulators were comparable with each other.

Discriminative: Entropy Metric provided real numbers and eliminating the risk of ties.

Generality: Being applicable to different crowd simulators.

Entropy Metric was used to compare simulators with each other as well[12]. Same simulators proved to be the most similar with each other which was the expected result and similarity of the simulators with each other was able to be calculated as well.

User Study

A user study was conducted with 36 participants (22 male) as well to compare the Entropy Metric with the visual perception[12]. According to the results, when Entropy score was greater than 0.1, the metric predicted the user response correctly. When the relative differences in Entropy scores between two simulators were less than 0.1, metric failed to correctly predict user preferences at a statistically significant level. Though the metric classified the simulators as "very similar" with simulators classified as similar by the users as well[12].

2.11.2 Path Patterns

Path Patterns is an evaluation method that evaluates the crowds by extracting their trajectories and using a Bayesian model and it was proposed by Wang et al.[41]. First step of this method is to find a set of paths which is a collection of similar trajectories. A non-parametric hierarchical Bayesian model is formed to compute the distributions of these trajectories. The dataset from the crowds are then divided into training data and a test data segment. Model is then trained for the given training data.

Similarity metric can be calculated between two algorithms or real data. Then the metric calculates the likelihood of algorithm B given algorithm A. This is due to the usage of a Bayesian model.

Results varied from model to model and did not give good results for all the test cases[41]. It estimated paths which only a few pedestrians took, for example. Wang et al. also stated that converging this metric was time consuming. In one instance, after waiting for 40 hours, method has not reached a final value for the metric.

2.11.3 Context-Dependent Crowd Evaluation

This method is using footage of real crowds and generates a database of behaviour examples[23].

Idea of Lerner et al. is to perform an analysis given a crowd simulation by defining a set of queries. Results of the analysis can be used to decide if the given crowd simulator is similar to the real crowd.

Lerner et al. a framework to extract the trajectories of the pedestrians in a given video. These trajectories were then put through a simulator to generate the virtual crowd.

Evaluation of the crowds were divided into two: Short and Long term evaluation. For the short term evaluation, two pedestrian measures, density and proximity, were used. Density is the number of pedestrians surrounding the individual agent. Proximity is used to store the distance to the nearest person within the sectors around the individual agent. For the flock behaviour, directions of the agents that best matches the direction in the sectors towards the subject agent.

For the long term evaluation, changes in the trajectory of the agents were taken into consideration. Changes in direction and speed in the long term trajectory of the agents were calculated. Also, these were separated based on whether agents had companions with them or not since it affects the walk pattern of the virtual agent.

For the short term evaluation, calculated similarity rates gave accurate results i.e, if the score for an agent was low, it meant that the agent was behaving unnaturally which was the case for the said agents.

Results for the long term evaluation was indecisive. Lerner et al. stated that the generated results were not enough to make an assessment about the generated crowd.

2.11.4 SteerBench

Overview

SteerBench is a framework for evaluating the generated virtual crowds and proposed by Singh et al. [36] and their work is explained below. It uses set of steering behaviours and test cases to evaluate the generated virtual crowds. Based on the displayed behaviour of the virtual crowd algorithm, SteerBench gives it a rank.

SteerBench has two major parts. The first one is a set of test cases and the second one is a benchmark evaluation approach for computing some metrics and scores for the steering algorithm.

Generated scores from this framework does not indicate an algorithm being better than another. For each test case, the user needs to look at the generated scores and decide if the algorithm produces

acceptable results unlike Entropy Metric where the generated score claimed to be an indicator of a good or bad algorithm in terms of behaviour of the agents.

Test Cases

There are 38 scenarios currently in SteerBench[36]. From these scenarios 56 test cases are derived. These test cases are classified under five categories.

1. **Simple validation scenarios**: These scenarios tests very simple scenarios that every algorithm should be able to handle.
2. **Basic one-on-one scenarios**: Tests two agents to steer around each other without obstacles
3. **Agent interactions including obstacles**: Tests an agent's ability to steer around other agents with the presence of obstacles.
4. **Group interactions**: Tests the ability of the algorithm to handle common situation occur in the real crowds. Has a group of agents and several obstacles.
5. **Large-scale scenarios**: These test cases are done to stress test an algorithm.

Metrics for Evaluation

There are some metrics that are calculated with SteerBench. This metrics are divided into categories[36].

1. **Primary Metrics**: This metrics are the important metrics that agents should get a high score. Because natural crowds avoid these situations such as colliding with other pedestrians.
    - *Number of Collisions*: Collisions should be avoided. Indicates poor behaviours if this number is too large.
    - *Time Efficiency*: Time efficiency is defined by how fast the agent in the simulation is able to reach to its goal. It is measure as the total time an agent spends to reach its goal.

- *Effort Efficiency*: This metric is for measuring how much effort and agent makes to reach to the goal. This is measured as total kinetic energy that an agent uses to reach its goal.

2. **Detailed Metrics**: SteerBench has many others defined metrics for the steering behaviours. These include distance and speed change as well as acceleration and turning rates. They are useful to get a more detailed information about the tested algorithm as well as analysing the algorithm in more detail for given scenarios.

Each test case defines several constraints. These constraints are defined to check if there are obvious errors within the algorithm. These constraints are reaching the goal, zero collisions and in case of some test cases, reaching the goal within a certain amount of time.

Calculating the Scores

A score can be calculated for a single agent, all agents or for all test cases. Scores are calculated for a single agent, for all agents or for all test cases.

1. **Score for Single Agent**: Score for a single agent is calculated by combining the primary metrics. Each test case defines different weights and each agent has different weights.

2. **Score for All the Agents**: Averages of the primary metrics are computed over n agents and weighted sum is computed with additional set of weights specific to the test case. This enables the user to define custom weights for the agents.

3. **Score over All Test Cases**: A sum of all scores from each test case is computed to get this score.

SteerBench allows the user to customize the algorithms defined along with the test cases and the weights for the agents.

Q Learning, RVO and Data Driven crowds were used during the evaluation of the methods. Q Learning was implemented and RVO was used as a library.

Chapter 3

Implementation

This chapter will describe how states and actions were represented using the work presented in [7], how the rewards for the actions were calculated, what the different reward calculations were used for virtual crowd generation, how the agents were trained and how the crowds were generated for the Q Learning with ϵ-greedy policy. Then it will describe how the RVO library was used for crowd generation.

3.1 Q-Learning

The algorithm for the Q Learning can be seen in algorithm 1.

Algorithm 1 Q Learning Algorithm

```
1: Initialize Q(s, a) arbitrarily
2: repeat for each episode
3:     Initialize s
4:     for each step of the episode do
5:         Choose a from s using ε-greedy policy
6:         Do action a and observe r and s'
7:         Q(s, a) ← Q(s, a) + α[r + γ max_α' Q(s', a') − Q(s, a)]
8:         s ← s'
9: until s is terminal
```

In the algorithm 1, s is the state the agent is in, a is the action taken for the current state and r is the reward for the action. Variable α is called the learning rate. Learning rate determines the rate of overriding the old information with the new one. Variable γ is called the dis-

count factor. This factor determines the rate of which the new rewards will be considered. For example, if this factor is 0, then the agent will only consider the currently received rewards. Both α and γ is set to 0.1. A large value for α is not wanted since it will make the agent consider only the new information. Therefore, α was set to 0.1. Setting γ to 0.1 makes older values more important. This is needed since only using the new rewards will not result with accurate results since old rewards will not be taken into account. Selection of the actions was done with the ϵ-greedy policy since it was stated in [7].

The agent needs to observe different states and receive rewards for them. This process is called "learning". When an agent finished learning, it's experience can be used by the other agents.

Creation of crowds was done in two stages. First stage was the learning stage and the second stage was the agent creation stage.

3.1.1 Learning Stage

At this stage, there is only one agent in the simulation and at each frame, agent does 100 learning iterations. Obstacles are static with radii ranging from 0.1 to 1.5 and there is only one goal. Number of obstacles are 450 since more obstacles will enable the agent to experience more states because agent will be in different states more often due to the high number of obstacles around the agent. States, actions and the rewards will be described below.

State Calculation

An agent has two states: obstacle state and goal state. In order to calculate the obstacle state, an occupancy code was defined (see figure 2.9). Occupancy code is an 8 digit number and it was retrieved and converted into a single number for calculating the obstacle state.

This occupancy code was filled with numbers according to the distance from the agent's position to the obstacles around it (see figure 2.9 in section 2.8.1).

Goal state was calculated with built in Unity Engine functions. Agent had two different states depending on whether it reached its goal or not. A method was written to separate the two cases (see Algorithm 2).

Obstacle State Obstacle state for the agent is the conversion of the occupancy code into a single number. This was done by using the for-

Algorithm 2 Current State of an Agent

1: **if** Agent is at goal position **then**
2: $(numDistances + 1)^{nrOs} * numGoalState$
3: **else**
4: $(numDistances + 1)^{nrOs} * goalState + obstacleState$

mula given below. Built-in functions of the C# programming language was used for the calculation of this number.

$$obstacleState = \sum_{i=0}^{nrOS-1} OccupancyCode[i] * ((nrdistances + 1)^{nrOS-1-i}) \tag{3.1}$$

Populating the occupancy code at the current stage of implementation requires more computational power. If the distance is calculated from the agent to the closest obstacle, occupancy code will not be filled accurately since distance is calculated from the center of the agent to the center of the obstacle. Main problem here is to see how large is the area of the obstacle inside the agent's radius of vision.

Raycasting method was used for this purpose. Rays served as a sensor for the agent and were created from the agent's location to outward direction from the agent's center. Length of a ray is the maximum distance that the agent can perceive(see figure 3.1).

It was then possible to fill in the occupancy code accurately since it was now known exactly where there were an obstacle (see algorithm 3).

Algorithm 3 Obstacle State Calculation

1: **for** i = 0 to 360 **do**
2: cast a ray from position of the agent towards forward direction rotated *i* degrees
3: **if** ray hits an obstacle **then**
4: Calculate hit distance
5: Fill in the occupancy code

Goal State Goal state is calculated by taking the angle between the agent's forward facing direction and the direction from the agent's position towards the goal position. Goal states are divided into intervals.

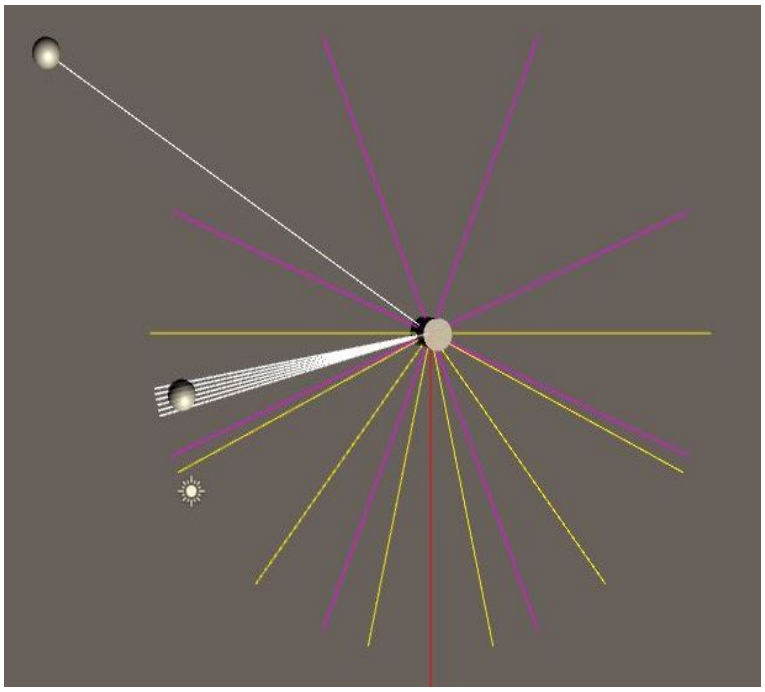

Figure 3.1: Obstacle state calculation. While lines are the rays that hit an obstacle and the long white line is drawn from the agent to the goal. Purple lines are the intervals for the actions the agent may take and the yellow lines are the intervals for perception of obstacles.

There were 8 goal states. Returned angle was compared and according to the interval it fell, that interval number was returned as the goal state.

In order to calculate the goal state, an angle between 0 and 360 was needed. Unity Game Engine does not provide a support for this. In order to get around this problem, agent's facing direction was calculated as well in order to control if the goal was behind the agent. If the goal was behind the agent, calculated degree was subtracted from 360 to get a result between 0 and 360.

Taking an Action

A separate method was implemented for taking an action. Casadiego et al. defined a unit of movement as 0.03. This was the case with the actions. After an action was selected, corresponding angle to rotate as degrees was taken. First, agent was rotated according to this angle and moved for 0.03 units (see figure 2.11).

Calculation of the rotation of the 3D pedestrian model was done separately from the rotation of the agent because agents were changing rotation in a rate which is can not be witnessed in real world (6 times in time elapsed between one frame and the next). Rotation of the agent was changed to the direction of the movement. Rotation was then updated in each 100th frame because when this number was lower than 100, agents rotated almost at the same rate they did before and for the number higher than 100, rotation happened after the direction of the agent changed.

Calculating the Reward

This is the most important part of the algorithm since giving correct rewards for the states will define how the agent will behave in the environment.

Several approaches were tested for calculating the obstacle reward. Two approaches which provides a collision avoiding behaviour was selected for evaluation.

Goal reward is calculated by taking the cosine between the direction vector of the agent and the direction towards the goal as stated in the background section. Casadiego et al. stated that they have tried another approach for calculating the goal reward yet taking the cosine between the direction vector of the agent and the direction towards the goal yielded smoother trajectories and agents reached their goals faster. Therefore, goal reward function was not changed for the two different reward approaches.

Two different reward calculation approaches selected are given below. For the equations given, $nrOS$ is the number of Obstacle States and $DOC_i = occupancyCode[i]$.

First Reward Approach Overall reward is given according to the formula below.

$$reward = \begin{cases} rewardGoal & \text{if there is no obstacle} \\ rewardObstacle + rewardGoal & \text{otherwise} \end{cases}$$

Agent is still rewarded if there is an obstacle in the radius of vision. Rewards for detecting obstacles was calculated with the equation below.

$$rewardObstacle = \begin{cases} 10 * rewardGoal & \text{if } DOC_i = 0 \\ -\sum_{i=0}^{nrOS-1} 10/10^{(DOC_i-1)*2} & \text{otherwise} \end{cases}$$

Second Reward Approach This approach has similarities with the first approach regarding the calculation of the obstacle rewards. Total reward is calculated differently.

$$reward = \begin{cases} rewardGoal & \text{if there is no obstacle} \\ rewardObstacle & \text{otherwise} \end{cases}$$

As can be seen, agent is not rewarded with the closeness to the goal if there is an obstacle. This tells the agent to put more emphasis on avoiding the obstacle rather than going towards it. Reward for obstacles is given below.

$$rewardObstacle = \begin{cases} 100 * rewardGoal & \text{if } DOC_i = 0 \\ -\sum_{i=0}^{nrOS-1} 10/10^{(DOC_i-1)*2} & \text{otherwise} \end{cases}$$

Learning Iteration

A single learning iteration refers to calculating a state, taking an action, getting the reward and updating the Q Learning table. Number of iterations are defined as 1.5 million. Selecting an appropriate number for the number of iterations was done by using the resulting Q values and checking whether the agents were able to avoid collisions while reaching their goals. Also, size of the Q table and whether a state had changed values for the actions were investigated in order to find the defined number of 1.5 million iterations.

After defining the states, actions and rewards along with Q Learning algorithm, teaching an agent about the environment is done with algorithm 4. It is at this stage, goal position and the obstacle positions are randomized within the boundaries of the current scene.

When an iteration is finished, agent is set to its start position and the Q Table is written to the disk as a plain text file where all the values are separated with commas. Reason for this file format was to make saving and loading the Q Values as simple as possible since reading from the disk and writing to it would only be done once while agents were being generated for crowd simulation and need a Q table for taking actions. These text files were then saved in the local directory for later usage and named to indicate which reward functions were used.

To create the crowd simulation generated text file was loaded into the memory for a crowd simulation. Generated agents only calculated

Algorithm 4 Q Learning Agent Learning

1: **repeat**for each episode
2: **if** Agent is at goal position **then** randomize obstacles and the goal positions
3: **if** Number of iterations are divisible by 10000 **then** randomize the goal position
4: do a learning iteration
5: increment iteration number
6: set occupancy code to zero
7: **until** Iterations are finished

the state they were in. According to the state number, they chose the highest Q value from the table and do their action accordingly.

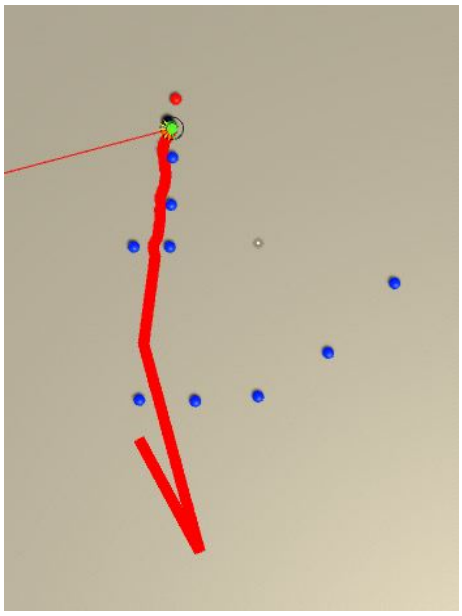

Figure 3.2: Result after training the agent with the final reward approach. Blue spheres are obstacles. Red sphere is representing the goal location.

3.1.2 Agent Generation Stage

In the scene, there is an empty game object that contains a script for generating the virtual crowd agents. It generates the agent and saves

the resulting Q Tables. Each agent has a script for doing the learning iterations and setting up the test cases. Detailed description is given below for these two scripts.

- **Agent Generator** This script generates agents to the scene. It first reads a Q Table from the disk and feeds the table into the newly generated agents for them to use. Main purpose of this script is to create different scenarios for the actual virtual crowds. Several generation coordinates exists for this method. For example, in order to create agents around a circle, first it reads the Q values and stores them, gives the goal positions to each agent then it initializes agents and gives them the current Q table.
- **Q Learning script** This script serves as the brain of the agent it is attached to. Q Learning algorithm described above is implemented in this script. State and reward calculations along with randomizing the obstacle and goal positions are also done in this script. At the end of all the iterations, Q Table is written to the disk at this script as well. Moving according to a Q Table is also done in this script.

3.2 RVO implementation

3.2.1 Overview

Currently RVO library is available in C++ and C# programming languages and support 3D and 2D usage. Since Unity Game engine supports C# better, C# version is used for the implementation. There was no elevated surface in the simulation. Therefore, it was possible to use 2D version of the library. Only thing to consider was to set the values for the Y axis to a fixed value (0 in this case). Remaining parts of the implementation was done according to the example provided in the documentation of the library.

A simulator instance needs to be present which contains all the locations of the agents in the scene along with obstacles. In each time step, this simulator updates the locations of the agents. Locations are then sent to the agents in the scene. They move accordingly and the new information is fed to the simulator instance again.

3.2.2 Agent creation

There were several parameters to consider before creating the agent. It is possible to create agents with different radius of collision, speed and acceleration. Their position and goal needs to be given as they are initialized. Algorithm for creating agents is given in the algorithm 5.

Algorithm 5 RVO Agent Creation Process

1: **repeat** for each agent
2: calculate agent's start position
3: calculate agent's goal position
4: Add agent position to the simulation
5: Save the agents position
6: **until** all agents are created

After the creation of the agents are completed, next step was to update their positions at every frame. A method similar to this one has already been implemented and provided to the RVO samples. Algorithm for updating the velocities of the agents in the simulation is given in algorithm 6.

During the initialization of the agents, deadlocks may occur. Deadlock means in this context for each agent in the simulation is not being able to move since the movement depends on the other agent's movement. Since other agents can not move, no movement is done by the agents. In order to avoid this, start positions of the agents needed to be changed Either during the simulation or at the start, if the positions of the agents reached to a convergence before the end of the simulation, position of an agent was randomized.

Algorithm 6 RVO Update Velocities

1: **for** each agent inside the simulation **do**
2: Calculate direction to the Goal
3: **if** magnitude of the direction to the goal> 1.0 **then** Normalize the vector
4: Update the agent's velocity according to the Goal direction
5: **if** Agent is at the goal position **then** Remove the agent

3.2.3 Rotating the Agents

RVO2 library does not provide a rotation support for the agents. Since only their positions are updated, it is necessary to change the rotations of the agents as well. Rotations of the agents were changed according to the direction they were heading.

After the agent positions were updated, direction was calculated relative to the agents position in the previous frame. Using the Unity's built in functions, the agent facing direction was changed accordingly.

Yet this approach only provided valid results when the position of the agent was not changing rapidly. If the agent was moving towards different locations rapidly, then the rotation was changing rapidly which may have affected how the user will perceive the paths of the algorithms and the crowds generated with them.

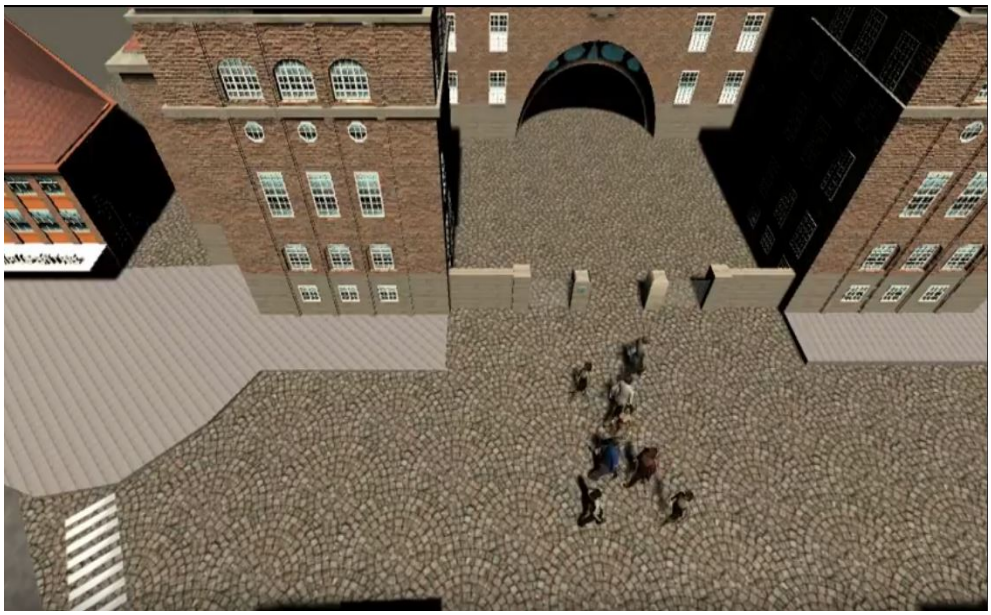

Figure 3.3: A crowd generated with RVO

This problem was solved with the following method. Position of the agent in each frame was stored in a list. At every 100th frame, the rotation of the agent was changed by calculating the direction from the agents position at the 100th frame, taken from the list, to its current position during the current frame. After rotating the agent, list was updated with the new position of the agent. This approach ensures that the rotation will not change rapidly even though the agent rotates

rapidly. Figure 3.3 shows a crowd generated using RVO in Unity Game Engine.

3.3 System Specifications

Thesis was implemented in free version of the Unity Game Engine. Scripting language was C#. System had Intel Core i7 2.80 GHz and Nvidia GTX 870M GPU.

Chapter 4

Evaluation

This chapter will give the results of the user study. Analysis of the results will be presented along with the responses to the questions.

4.1 Overview

Evaluation of the generated virtual crowds were done with a user study. In the study, main objective was to investigate how crowds generated by Q Learning was perceived. Two different cases was tested. Details about the study is given below.

4.2 User Study

Participants for the user study was selected from KTH university students ages ranging from 23 to 26 with 7 participants (6 male, 1 female). Crowds were shown in the form of videos. Pedestrians were assets from the Unity Game Engine asset store [1]. Crowds generated from real crowd data were also provided by two students who implemented the method for their bachelor thesis work [24].

4.3 Variables

Variables are the factors that results with different trajectories and behaviours for the crowds. Below, the variables are given.

4.3.1 Reward Approaches

Different reward functions are described in the implementation chapter. In order to have more information about the perception of the Q Learning method, the reward functions that had collided and went through each other was omitted. As a result, only two different reward approaches remained. First reward approach is rewarding the open intervals positively. Second approach is the approach that also rewards open intervals positively and not calculating the goal reward when there is an obstacle within the perception radius of the agent. From this point on, they will be referred to as first and second reward approach respectively. See section 3.1 subsection "Calculating the Reward" for a detailed explanation of the rewards.

4.3.2 Collision Detection Distance

RVO library provides a setting which is to set the distance when the collision avoidance will start. This is an integer number and if it is set too low, simulation will not be safe (stated in the documentation of RVO library) and there may be deadlocks where every agent is waiting for the other agent to move. Changing this number gave different results with the simulations using RVO library. Low collision detection distance was 2 units and high collision detection distance was 5.

Same effect was done by setting the radius along the agents that are using Q Learning. For low collision detection, outer radius of the agent was 2 units and inner radius was 1.5 units. For high collision distance, outer radius was 5 and inner radius was 2.5 units long.

4.3.3 Scenes

There are two scenes, each described in detail below.

Trails

In this scenario, agent models are not shown. Instead, the path they followed was shown to the users. Reason for this approach is the rotation of the agents. With Reinforcement Learning approach there were cases where the agents were rotating rapidly. With this approach, users were able to concentrate on the paths of the agents and will not be bothered with other factors at the scene see figure 4.1.

Figure 4.1: (left)Initial state of the agents (center) Agents status when they come close to the center (right)Agents when they move away from the center

KTH Scene

This test case was done with the KTH's entry where D and E buildings were present. This test case was created to test whether generated crowds would look realistic or not. There are 10 agents in total in the scene divided equally to opposite sides. Their goals are at the opposite side of their position (see figure 4.2). A video from Data Driven Crowds was used here for comparing the existing methods with real crowds.

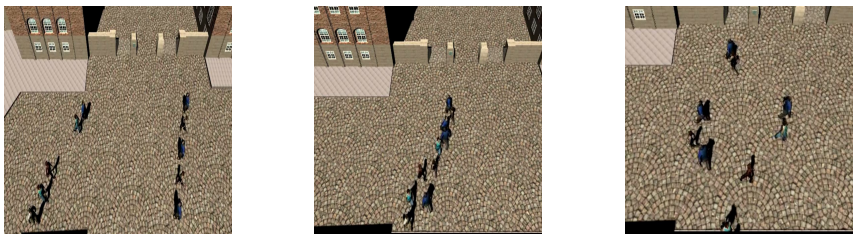

Figure 4.2: (left)Initial state of the agents (center) Agents status when they come close to the center (right)Agents when they move away from the center

4.4 Design of the Study

Two different sets were then formed. They were KTH scene and Trail scene. For each of these scenes, different collision detection distances and reward approaches were applied and resulting crowds were recorded. At KTH scenes 10 agents were used. Trail scene was recorded with 2 agents.

With every variable videos were recorded which in total was 6 videos. For KTH scene, data driven crowd approach was used so there were 7 videos for KTH scene. Each video was 10 seconds in length.

For each scene, 3 blocks were created. Each block had videos of the crowds generated with Reinforcement Learning, RVO and, for the KTH scene only, resulting crowds from the implementation of Malmström et al. [24] was used as stated in section 1.2. In each of these blocks, order of the videos were randomized. In total there were 57 videos with 10 seconds each summing up to 9.5 minutes. Order of the videos in the blocks were randomized, yet the order of the scenes were the same. After a training video, first the Trail scene videos and KTH scene videos were shown to the user.

4.5 Conducting the Study

Several heuristics were followed with the study. Most important aspect of the study was the duration of it. In order not to tire the users a maximum of 40 minutes were set. Another heuristic was the importance of the KTH scene. It was the scene where the data driven crowds could be used as well and it was the scene most similar to a real life situation. Therefore, it was more important for the analysis of the methods to get more results from the KTH scene.

Study started with a briefing to the user and signing of a consent form. Then a sample video was shown. This video was just to give an idea to the user what they were going to see. Then blocks were shown to the user. Users were handed an answer sheet for answering the questions. Questions asked for a video in the user study is given in the figure 4.3.

VIDEO 1	1-Not at all	2-Not much	3-Neutral	4-Somewhat	5-Very
How natural is the paths pedestrians followed ?					
How optimal was the paths of the pedestrians i.e How do they get to where they are going: Efficient or wasting energy					
How goal oriented was the pedestrians i.e Where do they seem to be going - have a specific destination or wandering aimlessly ?					
How natural does the crowd look overall?					
Did any of the characters walk through each other ? (YES or NO)					

Figure 4.3: Questions asked for a video during the user study

Each question is about the perception of the users about the scene and not about what actually happened in the scene. For example, it is important to know if the user spotted a collision than agents actually colliding and not being spotted by the users.

These questions were the same for each scene. Every time a user finished a block, a new question sheet was handed to the user. After the user stopped watching the videos, users were debriefed and experiment ended.

Scaling of the question were done according to the Likert scale. 1 is "not at all", 2 is "not much", 3 is "neutral", 4 is "somewhat" and 5 is "Very"

4.6 Results

This section will present the results gathered from the user study conducted.

4.6.1 Overview

The main goal of this study was to investigate whether using the Reinforcement Learning algorithm provided a better result or a realistic result. With this idea in mind, several aspects of the algorithms were selected in order to investigate the Reinforcement Learning algorithm with more detail and make an accurate assessment.

These aspects were the radius of the agents, distance allowed between two agents and their detection radius.

Regarding the graphs, abbreviations were used to display the different algorithms and methods used. Abbreviations can be seen below and should be used to read the graphs.

	Low Collision Detection	**High Collision Detection**	**First Reward Approach**	**Second Reward Approach**
Reinforcement Learning	Low	High	R1	R2
RVO	RVO Low	RVO High		

Table 4.1: Abbreviations used in the tables

"Data Driven" in the graphs refers to work of Malmström et al. meaning virtual crowds generated using the data from real crowds.

Expected Results

Expected results from the user study will be presented in this section.

1. **Trail Scene** For this scene, it is hypothesised that RVO will prove to be perceived as the most realistic. Reason for this is because agents generated by Reinforcement Learning methods are not aware of each others positions in the world. Therefore, they avoid collisions as soon as they detect each other. RVO on the other hand is able to calculate a velocity that will not end with a collision. Paths generated are smoother than Reinforcement Learning method. Reinforcement Learning methods with high collision detection distance will not be perceived as being natural. With an early detection, they will start collision avoidance early and that will make the agents seem they are wandering aimlessly.

2. **KTH Scene** It is hypothesised that Reinforcement Learning with second reward approach having low collision detection distance will be perceived either as natural as the RVO or more natural than both variants of RVO for the crowds. This result is predicted to be the same for the paths generated. Reinforcement Learning with low collision detection distance is predicted to be perceived as more goal oriented than RVO's variants. Results for the optimal path is predicted to be the same with RVO's variants and the second reward approach with low collision detection distance.

   Crowds generated with high collision detection distance of Reinforcement Learning will not be perceived as natural. Reason for this assumption is because there are many obstacles that will be taken into consideration which will alter the paths of the pedestrians. This will alter their perception regarding their orientation and their paths where it will have a negative effect.

4.6.2 Analysis

Each user was shown KTH and trail scenarios and three participants were also shown the circle scenario. Participants were given an answer

sheet to put in their responses. Results from each user were gathered in a single file. For each block, average of the results were taken. Each result was subjected to ANOVA test to check whether it was possible to reject null hypothesis. For each result, null hypothesis was rejected according to ANOVA test.

Naturalness of the Path

This part of the analysis corresponds to the question: "How natural was the path the pedestrians followed". Answers to trail and KTH scene will be presented below.

Trail scene Perception of the paths followed by the pedestrians can be seen in the figure 4.4. Two variants of RVO had higher averages than all the variants of the Reinforcement Learning agents. RVO with low collision detection distance was perceived as more natural than RVO with high collision detection distance. Second reward approach for the Reinforcement Learning method had the highest value comparing to the other variants of the algorithm. Low collision detection distance for the Reinforcement Learning algorithms resulted with higher values yet, according to the scale they were neutral. High collision detection for the first reward approach had the lowest value.

KTH scene Perception of the paths followed by the pedestrians can be seen in the figure 4.5. Data driven crowds was perceived as the most natural paths comparing to every other algorithm. Both variants of RVO followed with high collision detection distance had a higher value. Second reward approach with low collision detection radius followed after them. Users perceived the crowd generated as neutral. Difference between the result for the trail path is around 0.2. Improvement was greater for the Reinforcement Learning algorithm with high collision detection distance. For the second reward approach, overall result was 2.4 for the KTH scene where for the trail scene this number was 1.69. Variants with low collision detection showed improvements as well but the difference was 0.1 for the low collision detection and 0.3 for the high collision detection.

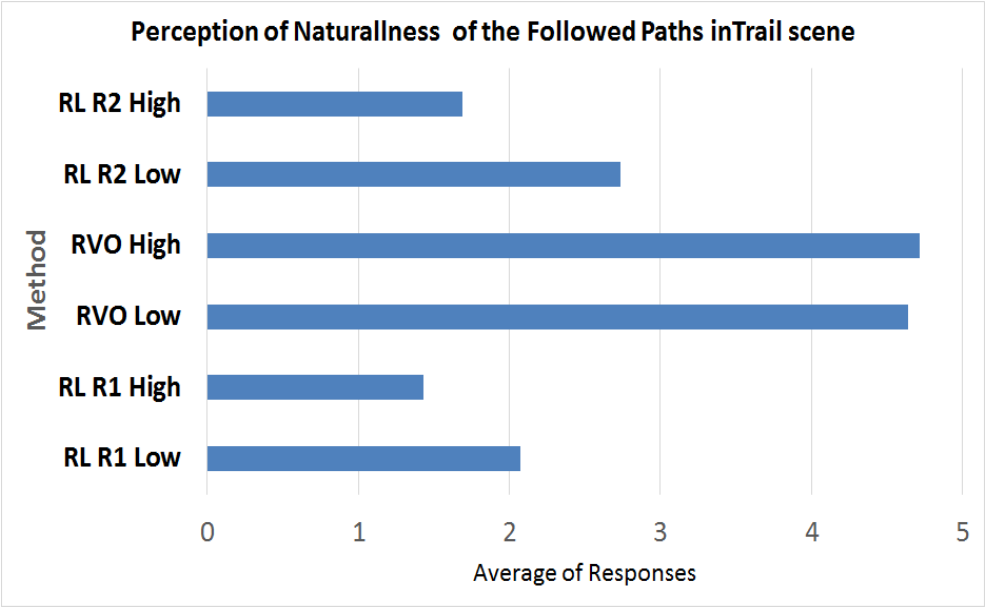

Figure 4.4: Perception of naturalness of the paths in Trail scene

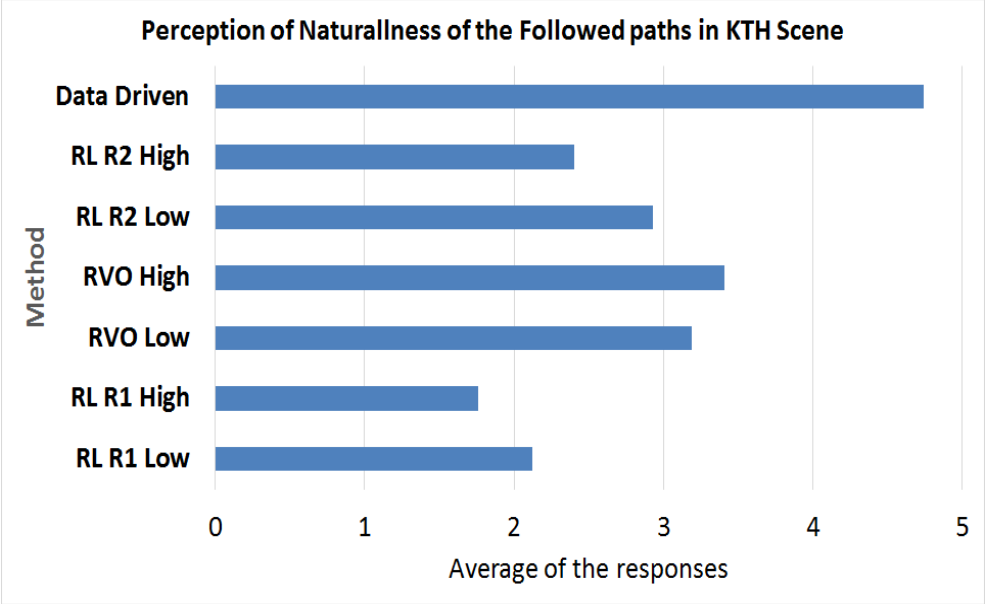

Figure 4.5: Perception of Naturallness of the Followed paths in KTH Scene

Naturalness of the Crowd

This part is the answer to the question "How natural does the crowd look overall?". Answers to trail and KTH scene will be presented be-

low.

Trail scene Perception of the naturalness of the crowds can be seen in the figure 4.4. Results for this question is not different from the results of the naturalness of the paths. Again RVO's two variants was perceived as the most realistic which was followed by the Reinforcement Learning approach with low collision detection and second reward approach.

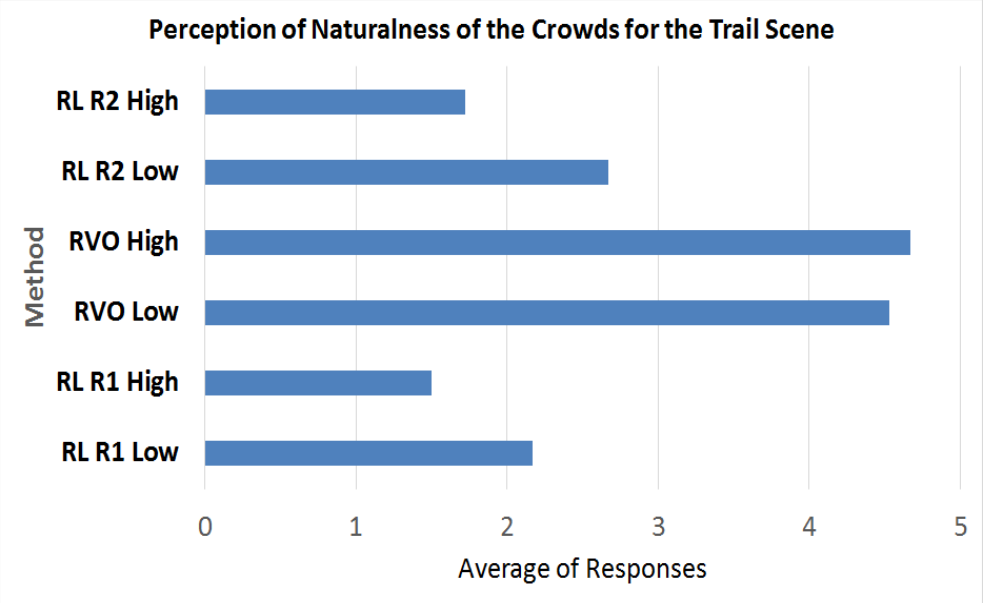

Figure 4.6: Perception of naturalness of the crowd in Trail scene

KTH scene Perception of the naturalness of the crowds can be seen in the figure 4.7. Similar to the trail scene, results for the KTH scene was similar with the naturalness of the paths. RVO with the lower collision detection distance was perceived as more realistic however with the value being 0.3 larger. Data driven crowd again had the highest value. Results for the Reinforcement Learning methods did not change.

Goal Orientedness

Being goal oriented means how the agents were going to where they were going i.e did they wander aimlessly or did it look like they have a

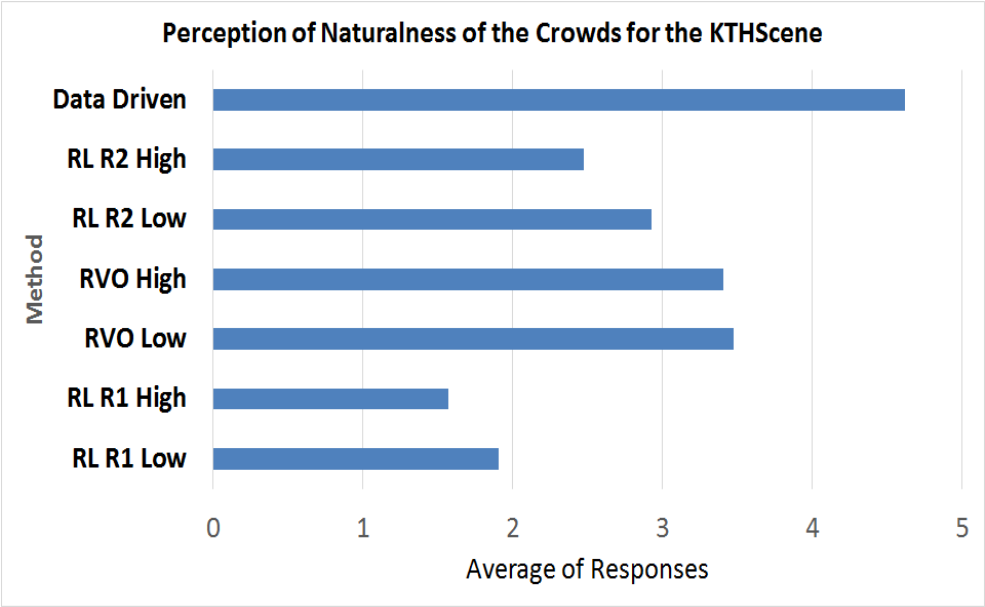

Figure 4.7: Perception of Naturallness of the crowd in KTH Scene

specific goal. Answers to trail and KTH scene will be presented below.

Trail scene Perception of the goal orientedness of the pedestrians can be seen in the figure 4.8. RVO's variants was perceived as being the most goal oriented. They were followed by second then first reward approach for Reinforcement Learning with low collision detection distance. Second reward approach with high collision detection distance and the first reward approach with the high collision detection distance had almost the same results and they had the lowest values.

KTH scene Perception of the goal orientedness of the pedestrians can be seen in the figure 4.9. For this part, second reward approach of Reinforcement Learning with low collision detection distance was perceived as more goal oriented than RVO with high collision detection distance and it was perceived as the same with RVO with low collision detection distance. Data driven crowds were perceived as the most goal oriented. First reward approach with low collision detection distance was perceived as being neutral and high collision detection variant had the lowest value.

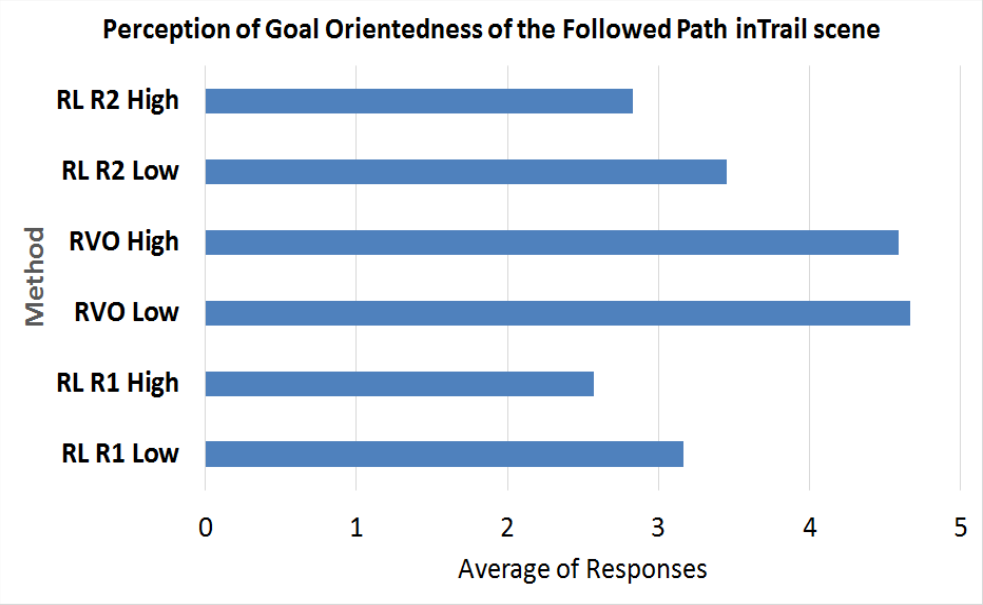

Figure 4.8: Perception of goal orientedness of the crowd in Trail scene

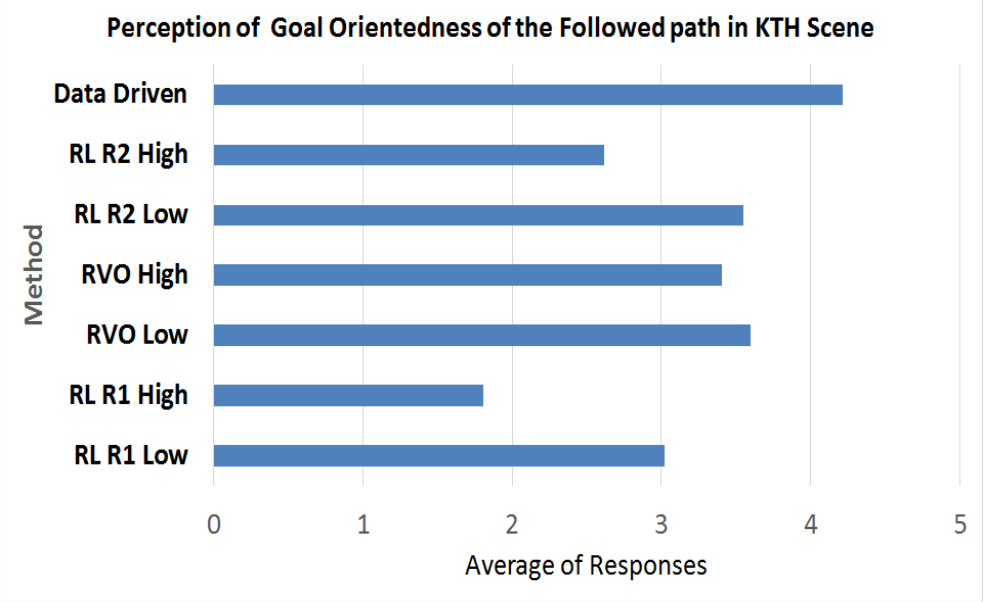

Figure 4.9: Perception of Goal Orientedness of the Followed path in KTH Scene

Path Optimality

By saying an optimal path, efficiency of the path is taken into consideration. This can be thought of energy efficiency for an agent. For

example, did the agent reached to the goal by taking a longer route of a more direct route. Answers to trail and KTH scene will be presented below.

Trail scene Perception of the path optimality of the pedestrians can be seen in the figure 4.10. Overall, results for this section is similar to the results for the trail scene. Meaning, RVO's variants has the highest value and they are followed by second reward approach with low collision detection distance for the Reinforcement Learning method. It is followed by the low collision detection distance for the first reward approach then high collision detection distances having the lowest values.

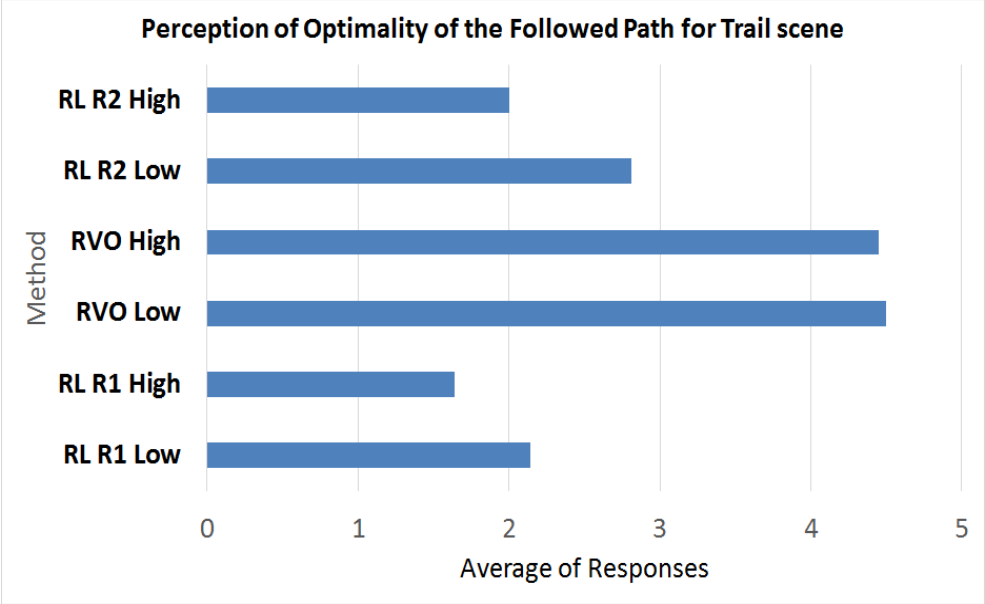

Figure 4.10: Perception of Optimality of the Followed Path for Trail scene

KTH scene Perception of the path optimality of the pedestrians can be seen in the figure 4.11. Results for this question is similar to the results for the goal orientedness of the paths. Results for RVO and Reinforcement Learning with low collision detection second reward approach is the same. Data driven crowd again was perceived as generating the most optimal paths. High collision detection for the first reward approach had the lowest value.

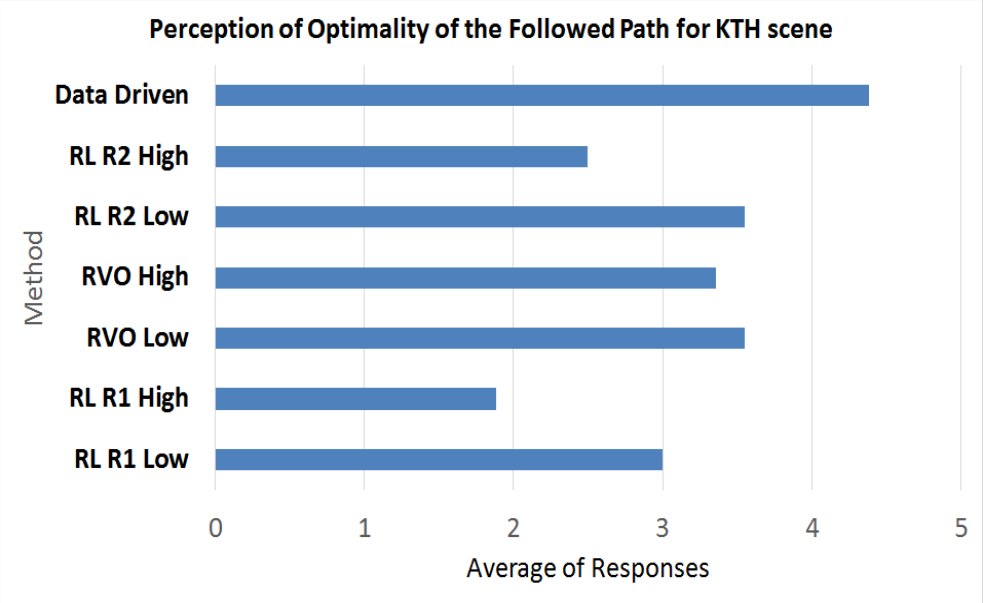

Figure 4.11: Perception of Optimality of the Followed Path for KTH scene

4.7 Analysis

Analysis of the results will be done in two sections. First section will be the analysis of the results from the Trail scene and the second section will describe the results from the KTH Scene.

4.7.1 Trail Scene Analysis

Overall, each algorithm scored consistently and scores did not vary from question to question. The worst performance was from the Reinforcement Learning algorithm with the first reward approach with high collision detection radius. It was followed by the high collision detection distance of second reward approach of Reinforcement Learning. RVO was the best performer by far where the average of the answers were given as crowds being the most realistic. Second reward approach with the low collision detection distance was perceived as more realistic than the other reward approaches and their variants. Yet, they were not close to RVO. For the question asking how natural the crowd looked, average result was 2,6 where for RVO with low collision detection distance, this number was 4.5 out of 5.

4.7.2 KTH Scene Analysis

For the KTH scene analysis, crowds generated from the data generated by real crowds were added. Crowd generated with this method was perceived as the most realistic one. Perception of the crowds generated by Reinforcement Learning resulted the same as the trail scene. Yet, average values in perception of the crowds was closer to 3 out of 5 which correspond to crowd being neutral. Second reward approach with low collision detection had the score of 2.9 for naturalness of the crowd. Perception of RVO however decreased compared to trail scene. For each question, perception both RVO with low and high collision detection radius was around 3.1 and 3.5. These numbers bring RVO closer to the Reinforcement Learning approach. This indicates that with an improved reward function Reinforcement Learning may be perceived as realistic as RVO.

4.8 Discussion

User study gave interesting results regarding RVO and the usage of Reinforcement Learning. One of the goals of this thesis was to evaluate the perception of crowds generated by this method which can create a basis for a discussion about the Reinforcement Learning method. These results show that we can use Reinforcement Learning for generating virtual crowds. Though according to the results, they are not more natural than RVO.

Copy-synthesis approach based on real-data was perceived as the most natural crowd as expected. Unexpectedly however, RVO method was perceived more realistic than second reward approach with low collision detection distance of Reinforcement Learning. Figure 4.7 and 4.6 shows that RVO generated more natural crowds. As hypothesised, when the collision detection radius increased, perception of the Reinforcement Learning methods decreased. Different from RVO, Reinforcement Learning takes into account more object around it and it not able to see the positions of the other agents. One more factor to consider can be the movement of the agents. When the detection radius increases, direction changes more rapidly even after dampening the rotation. This could have been a factor for the low values for the high collision detection radius.

For each different question in the user study, results for Reinforce-

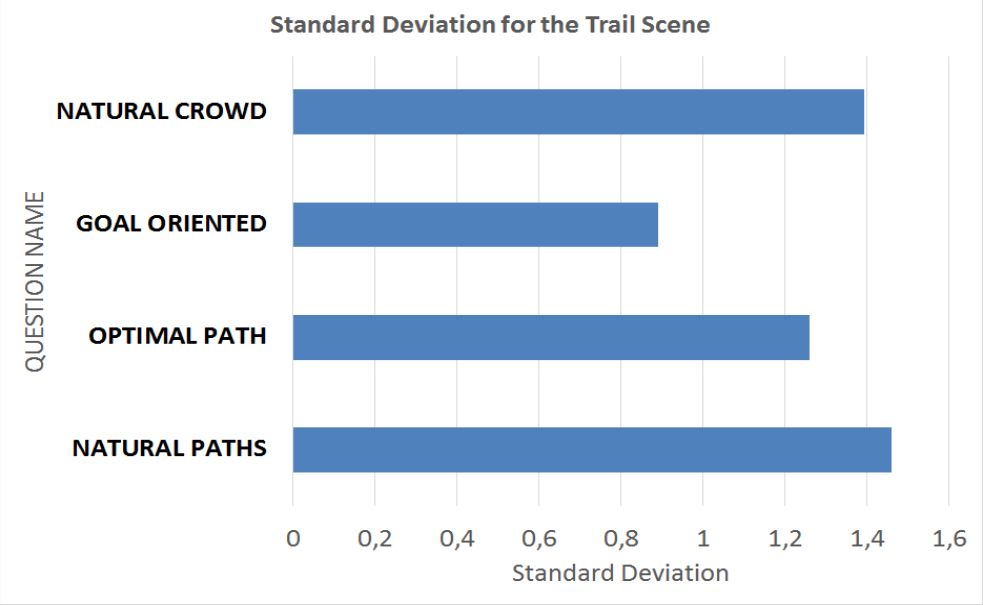

Figure 4.12: Standard deviation of the questions in the Trail scene

ment Learning and their variants were almost the same with each other. This is interesting to see. Also, these values for Reinforcement Learning stayed the same with KTH and trail scene and RVO's perception of naturalness decreased (see figures 4.7 and 4.6 for the KTH scene and 4.5 and 4.4). Trail scene and KTH scene did not have the same number of agents. Therefore, it can not be said whether the reason for this is the number of agents or the objects in the scene. There is however a correlation between these factors. As hypothesised, RVO was perceived as more natural goal oriented and having an optimal path than Reinforcement Learning. Reinforcement Learning was not perceived more natural than RVO contrary to hypothesis.

Second reward approach with low collision detection radius of Reinforcement Learning created more natural crowds from its variants. We can conclude that among the Reinforcement Learning variants low collision detection is the best one to generate virtual crowds.

Findings of this thesis shows that the approach of Casadiego et al is not able to generate natural looking crowds. It shows however that Q learning can be modified easily and new behaviours can be implemented.

Reinforcement Learning agents with low collision detection distance with second reward approach are more goal oriented and have a

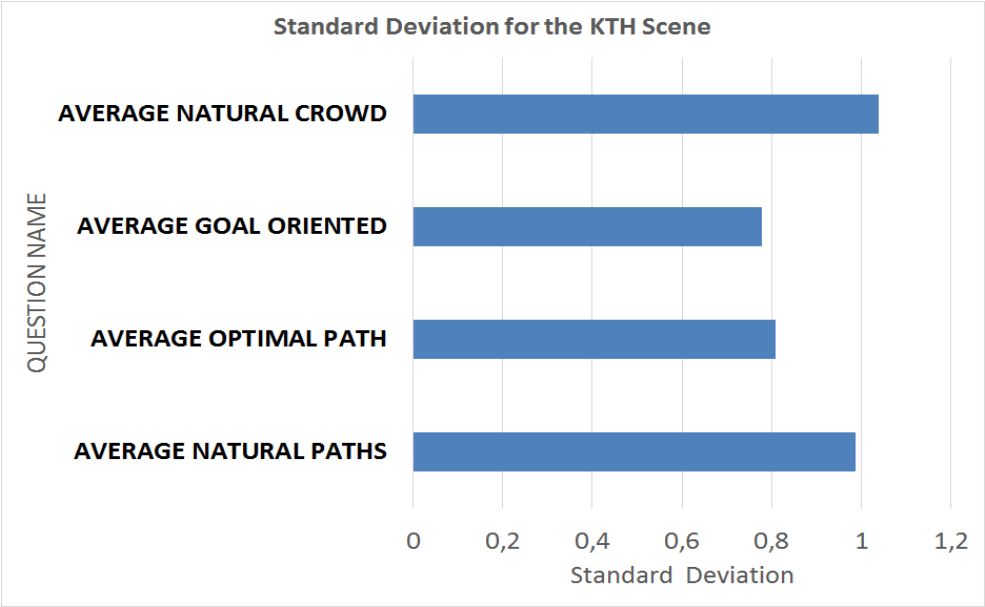

Figure 4.13: Standard deviation of the questions in the KTH scene

more optimal path than RVO in the KTH scene according to the results. Paths are not perceived this way for the trail scene. Users do not find the generated paths either natural or goal oriented when the see the trajectory. In a crowd simulation, just the paths are not shown to the user. Usually there is an environment around the agent. The best scenario to represent this was the KTH scene and the results shows that Reinforcement Learnings perception matches and passes the RVO's perception. Users said that Reinforcement Learning's second reward approach with low collision detection was more goal oriented and had a more optimal path which indicates that in a scene with objects and an environment, Reinforcement Learning may be used for creating a natural looking crowd.

Question asked in this thesis was whether Reinforcement Learning could be used to create plausible crowds comparing to RVO and real crowds. Comparing to real crowds, Reinforcement Learning needs more improvement. However, as stated before, preliminary results are promising. Comparing with RVO, it can be seen that Reinforcement Learning can be used to generate plausible crowds. Results from this study shows that Reinforcement Learning can be used for virtual crowd simulation.

Chapter 5

Conclusions

Goal of this paper was to investigate whether it was possible to generate virtual crowds using Q Learning that would be perceived realistic by the users. Using the states and the action described in [7] and two different reward calculation methods, two different versions of Q Learning method was generated. Virtual crowds were created in two different settings. RVO and a copy-synthesis approach based on real-data were created for evaluating the Reinforcement Learning methods against two different crowd simulation methods. A user study was conducted for evaluation. Results show that, while it is not possible to generate highly realistic results, it is possible to generate decent results with the simulations.

As it comes to the Q Learning method, for the trail scene, it is evident that generated crowds are not at all realistic and natural comparing to RVO. However, the results gets better when there are static objects such as buildings and roads in the scene. When that is the case as it is in KTH scene, perception of the generated crowds gets better. Increase in the Q Learning method is not high, yet decrease in the RVO's perception brings it closer to Q Learning method which is a good result. Reason for this being a good result is the fact that perception of the Q Learning increased and it performed the same with RVO.

Perception of Q Learning method is consistent within the different methods, RVO's perception is not. This shows that, Q Learning can be a more stable method for usage. Given its simple usage, this makes it a potential candidate for future improvements.

For the reward approaches, not rewarding the agent according to the goal position when there is an obstacle proved to be a better ap-

proach. According to the perception of the virtual crowds, second reward approach with low collision detection of Reinforcement Learning was perceived more natural than first reward approach with both low and high collision detection distance and second reward approach with high collision detection distance.

Crowds generated with Reinforcement Learning method are not perceived as natural as crowd generated with RVO. However, when the number of agents increase and in an environment with static objects such as buildings, perception of RVO becomes similar with Reinforcement Learning. Thus, we can conclude that Reinforcement Learning method can be used to generate plausible virtual crowds and these crowds can be perceived almost as natural as crowds generated with RVO.

5.1 Ethics and Sustainability

User participated to the study were assured that their privacy was to be protected and no information that could have led to them. Data acquired was anonymous and only the necessary information was asked from the users. In order to further protect their information, names of the participants were given a special code so in case another participants would have seen the answers given by another user, there would be no way of knowing who did those answers belong to. Nationality is not taken into consideration in this thesis as well further decreasing the personal data users had to give.

Q Learning method enables using Q values generated by one agent for other agents in the crowd simulation. With this way, complexity of virtual crowd generation can be reduced since agents will only have to read and select values from the given Q table. In RVO for example, velocities have to be selected for each agent in the simulation at each time step. With Q Learning, agents only will need a Q table for input and the goal location.

5.2 Future Work

5.2.1 Deep Q Learning

This option has variety of versions for different environments. None, to the knowledge of the author implements a crowd simulation with the Deep Q Learning approach. Reason for the possible investigation of this approach is because more states can be represented. For example, instead of dividing the agent's perception into intervals, possible moveable locations can be represented. Next step for testing this approach should be to implement this algorithm.

5.2.2 High Density Crowds

High density crowds refers to virtual crowds with many agents. An example to such crowds can be a central train station in a rush hour. Simulations with high density crowds may produce interesting results. Many agents in the environment can change the perception of the crowd as well. Making a user study with such crowds could provide interesting and may create another area of the crowd simulation where Q Learning can be used.

Bibliography

[1] *3DRT - Realpeople Females VR / AR / low-poly 3D model*. cgtrader. URL: https://www.cgtrader.com/3d-models/character/woman/3drt-realpeople-females.

[2] Hee Rak Beom and Hyung Suck Cho. "A sensor-based navigation for a mobile robot using fuzzy logic and reinforcement learning". In: *IEEE transactions on Systems, Man, and Cybernetics* 25.3 (1995), pp. 464–477.

[3] Aniket Bera, Sujeong Kim, and Dinesh Manocha. "Efficient trajectory extraction and parameter learning for data-driven crowd simulation". In: *Proceedings of the 41st Graphics Interface Conference*. Canadian Information Processing Society. 2015, pp. 65–72.

[4] Jur Van den Berg, Ming Lin, and Dinesh Manocha. "Reciprocal velocity obstacles for real-time multi-agent navigation". In: (2008), pp. 1928–1935.

[5] Ming Lin Van den Berg Jur and Dinesh Manocha. "Universal power law governing pedestrian interactions." In: *IEEE International Conference on. IEEE. Robotics and Automation* (2008).

[6] Lilian De Oliveira Carneiro et al. "Crowd Evacuation using Cellular Automata: Simulation in a Soccer Stadium". In: *Virtual and Augmented Reality (SVR), 2013 XV Symposium on*. IEEE. 2013, pp. 240–243.

[7] Luiselena Casadiego Bastidas. "Social crowd controllers using reinforcement learning methods." In: (2014).

[8] Jan Dijkstra, Harry JP Timmermans, and AJ Jessurun. "A multi-agent cellular automata system for visualising simulated pedestrian activity". In: *Theory and Practical Issues on Cellular Automata*. Springer, 2001, pp. 29–36.

[9] Paolo Fiorini and Zvi Shiller. "Motion planning in dynamic environments using velocity obstacles". In: *The International Journal of Robotics Research* 17.7 (1998), pp. 760–772.

[10] Golaem. *Golaem Assets: City*. [Online; accessed May 27, 2017]. 2017. URL: http://golaem.com/sites/default/files/images/assets/city.jpg.

[11] Golaem. *Golaem Software Logo*. [Online; accessed May 27, 2017]. 2017. URL: http://golaem.com/sites/default/files/Golaem-Logo100K.jpg.

[12] et al. Guy Stephen J. "A statistical similarity measure for aggregate crowd dynamics". In: *ACM Transactions on Graphics* (2012).

[13] Laure Heïgeas et al. "A physically-based particle model of emergent crowd behaviors". In: *arXiv preprint arXiv:1005.4405* (2010).

[14] Dirk Helbing, Illés Farkas, and Tamas Vicsek. "Simulating dynamical features of escape panic". In: *Nature* 407.6803 (2000), pp. 487–490.

[15] Dirk Helbing and Peter Molnar. "Social force model for pedestrian dynamics". In: *Physical review E* 51.5 (1995), p. 4282.

[16] Bing-Qiang Huang, Guang-Yi Cao, and Min Guo. "Reinforcement learning neural network to the problem of autonomous mobile robot obstacle avoidance". In: *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*. Vol. 1. IEEE. 2005, pp. 85–89.

[17] Brian Skinner Karamouzas Ioannis and Stephen J. Guy. *Crossing Flows*. Youtube, 2014. URL: https://www.youtube.com/watch?v=-bnAe0n70TI.

[18] Brian Skinner Karamouzas Ioannis and Stephen J. Guy. *Hallway simulation*. Youtube, 2014. URL: https://youtu.be/PRJ7Z1g7e5c?t=7s.

[19] Brian Skinner Karamouzas Ioannis and Stephen J. Guy. "Universal power law governing pedestrian interactions." In: *Physical review letters* (2014).

[20] Ioannis Karamouzas, Brian Skinner, and Stephen J Guy. "Universal power law governing pedestrian interactions". In: *Physical review letters* 113.23 (2014), p. 238701.

[21] Kang Hoon Lee et al. “Group behavior from video: a data-driven approach to crowd simulation”. In: *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association. 2007, pp. 109–118.

[22] Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. “Crowds by example”. In: *Computer Graphics Forum*. Vol. 26. 3. Wiley Online Library. 2007, pp. 655–664.

[23] Alon Lerner et al. “Context-Dependent Crowd Evaluation”. In: *Computer Graphics Forum*. Vol. 29. 7. Wiley Online Library. 2010, pp. 2197–2206.

[24] David Malmström and Stefan Kaalen. “An Investigation of an Example-Based Method for Crowd Simulations”. In: (2017).

[25] Ronald A Metoyer and Jessica K Hodgins. “Reactive pedestrian path following from examples”. In: *Computer Animation and Social Agents, 2003. 16th International Conference on*. IEEE. 2003, pp. 149–156.

[26] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), pp. 529–533.

[27] Volodymyr Mnih et al. “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).

[28] Rahul Narain et al. “Aggregate dynamics for dense crowd simulation”. In: *ACM Transactions on Graphics (TOG)*. Vol. 28. 5. ACM. 2009, p. 122.

[29] Jan Ondřej et al. “A synthetic-vision based steering approach for crowd simulation”. In: *ACM Transactions on Graphics (TOG)*. Vol. 29. 4. ACM. 2010, p. 123.

[30] ORCA. *ORCA Website*. URL: `http://gamma.cs.unc.edu/ORCA/`.

[31] Xue Bin Peng, Glen Berseth, and Michiel van de Panne. “Terrain-adaptive locomotion skills using deep reinforcement learning”. In: *ACM Transactions on Graphics (TOG)* 35.4 (2016), p. 81.

[32] Leandro A Pereira et al. “Emergency evacuation models based on cellular automata with route changes and group fields”. In: *Physica A: Statistical Mechanics and its Applications* (2017).

[33] Craig Reynolds. *Steering Behaviors For Autonomous Characters*. [Online; accessed May 27, 2017]. URL: http://www.red3d.com/cwr/steer/pathfoll.gif.

[34] Craig W Reynolds. "Steering behaviors for autonomous characters". In: *Game developers conference*. Vol. 1999. 1999, pp. 763–782.

[35] David Silver and Demis Hassabis. "AlphaGo: Mastering the Ancient Game of Go with Machine Learning". In: *Research Blog* (2016).

[36] Shawn Singh et al. "SteerBench: a benchmark suite for evaluating steering behaviors". In: *Computer Animation and Virtual Worlds* 20.5-6 (2009), pp. 533–548.

[37] William D Smart and L Pack Kaelbling. "Effective reinforcement learning for mobile robots". In: *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*. Vol. 4. IEEE. 2002, pp. 3404–3410.

[38] Massive Software. *Massive Crowds in Game of Thrones Battle of the Bastards*. [Online; accessed May 27, 2017]. 2017. URL: http://www.massivesoftware.com/images/game_of_thrones/BB_0952a_0610_Stills006_wm.jpg.

[39] Adrien Treuille, Seth Cooper, and Zoran Popović. "Continuum crowds". In: *ACM Transactions on Graphics (TOG)*. Vol. 25. 3. ACM. 2006, pp. 1160–1168.

[40] Jur Van Den Berg et al. "Reciprocal n-body collision avoidance". In: *Robotics research*. Springer, 2011, pp. 3–19.

[41] He Wang, Jan Ondřej, and Carol O'Sullivan. "Path patterns: analyzing and comparing real and simulated crowds". In: *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM. 2016, pp. 49–57.

[42] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[43] Christopher John Cornish Hellaby Watkins. "Learning from delayed rewards". PhD thesis. University of Cambridge England, 1989.

[44] Yijiang Zhang et al. "Online inserting virtual characters into dynamic video scenes". In: *Computer Animation and Virtual Worlds* 22.6 (2011), pp. 499–510.

[45] Kan Zhiqiang et al. “Simulation of evacuation based on multi-agent and cellular automaton”. In: *Mechatronic Science, Electric Engineering and Computer (MEC), 2011 International Conference on*. IEEE. 2011, pp. 550–553.