CS255 Robocode Implementation Using AI Techniques

Student ID: 1622228

ABSTRACT

This report looks at the implementation of AI techniques, specifically highlighting simple reflex agents, learning agents and goal-based AI using Robocode's API. The paper will be oriented towards the advantages and disadvantages of the different techniques in terms of effectiveness in robocode. Furthermore, several methodologies well-known in robocode will be shown as examples of the different techniques and how the hsck98 robot has been implemented with these. The arithmetic of the functions will also be briefly explained along with tests to see the progress of the robot in different settings. Any improvements will also be discussed.

1. INTRODUCTION

The robot's main components can be split into two parts: movement and shooting. The movement is mostly based on the reflex agent with goal as the robot needs to choose the best course of actions possible by evaluating the end-of-turn goal state. The shooting part is the learning reflex agent which chooses between two shooting modes for an enemy independently to its performance against others. Looking at figure 1, the actuators would be the pool of actions available to the robot, the sensors would be the built-in events that can occur to the robot. The question "what will it be like if I do action A" would be equivalent to the evaluators put in the robot.

These AI techniques were paired up with other existing works explained in section 2 and the robot's distinct functions are graded by facing it against robots in the sample package.

2. RELATED WORK

The movement strategy was based on the *minimum-risk* [1] movement plan. By referencing sample codes of robots that use similar game plans; such as *HawkOnFire* [4] or *Diamond* [5] (among many others), the implementation was carried out.

As for the robot's shooting mechanics, the models used were head-on targeting [3] methods (the most common and simple) and linear targeting [2] (slightly more complex but very effective against the sample bots).

Coursework submitted for CS255 2018.
Department of Computer Science, The University of Warwick, CV4 7AL, UK.
Report compiled January 16, 2018

•

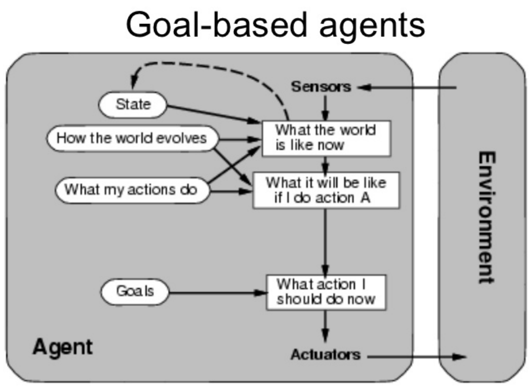

Figure 1: Goal-based agents in terms of the robot

3. STRATEGY DESIGN

Since the game's environment is discrete each move can be accounted for, so exact moves can be tailored and a large number of functions can be predetermined. Simple reflex agents perform exceptionally well in these types of environments where a set of condition-action rules can be applied.

3.1 Enemy Class

Just like many other robots, the hsck98 robot has an *enemy* class incorporated to store all the data required for our functions. Whenever the robot scans the area around it, if it scans an enemy the *onScannedRobot event* will activate returning the closest enemy. The *enemy* object initialization is implemented here because the robot wants to keep constant track of the opponents' status thus requiring frequent updates. But where is all this data saved?

3.2 Knowledge Storage

The robot's program architecture is based on a hashtable which is initialized at the start of the battle. It is re sizable depending on the total number of opponents and will remain empty until enemies are found during scan. The keys of the table are the names of the enemies, and the bucket values are the corresponding *enemy* objects defined earlier. The logic behind using hashtables is because it is simple to implement and very easy to traverse because of its unique feature *enumeration.* Enumeration provides a map of all the keys and values in the hashtable. It also allows iteration without the need of an iterator object.

3.3 Melee Movement: Minimum-Risk Movement Strategy

There are two fundamental segments of minimum-risk movement. Firstly, the random-point generator and secondly, the risk function. In AI terms, the random-point generator would be the different goals that we can achieve and the risk function would be considered the evaluation function to decide how good/bad a goal is and whether to take that course or not.

Before going on about how these work, clarification of why these were chosen is much needed. In melee situations, it is very difficult to keep an offensive stance since the robot can only shoot at one target at a time and because of the rules of the game, it is impossible to kill an enemy with full energy with just one bullet.

On the other hand, it is much more achievable to maintain a defensive stance because one can keep track of all enemies every time a scan is done thus avoiding the fights. This accomplishes two things: it allows our robot to survive longer (thus getting more survive points), and also makes enemies fight against each other therefore giving the robot an advantage during combat since the enemies will be at a lower energy.

The random-point generator is a simple function which by using the *Math.random()* function it selects an x and y coordinates within a given distance from the robot's current location. We make the function generate a large number of points (in our case 200), and then pass each point through the risk function to evaluate it. If the point generated is better than the last point one, the final(best) point is updated. Once all coordinates are assessed, the final point remains and the robot moves towards such destination.

The risk function is more complex. It incorporates the *Anti-Gravity movement theory.* It essentially obtains the current coordinates of the enemies and multiplies them by a constant; the inverse of the distance between the robot and the enemy squared (1/(d*d)). This assigns a force on each enemy depending on how close they are to the robot, similar to gravitational forces.

It is effective to not only have one single indicator for the evaluation as sometimes more distant enemies with a lot of energy can pose a bigger threat than closer enemies with low energy. In order to account for this, the energy of each enemy is inputted as well to calculate a ratio of enemy energy divided by the robot's energy which will indicate how dangerous an enemy is.

Lastly but not least, an indicator for the direction of the point is provided. What this means is that the angle between the robot and the designated point is obtained, as well as the angle between the enemy and the point. By subtracting these two the difference in angles is obtain and thus the bigger this angle is, the better off the robot is in going that way since the enemy will have to traverse more to catch up to it.

3.4 1v1 movement: Dodging Bullets

Although minimum-risk movement does not necessarily perform badly on 1v1's, it is true that it loses a bit of its purposes as it can easily be cornered since no wall avoidance is included. So in these cases, another game plan is carried out. By having the robot constantly adjusting its heading to the enemy's location, it can stay perpendicular to the target. By doing so, it assists the process of dodging the bullets by moving backwards and forwards only.

Although the environment is partially observable since bullets are not detected by the robot, it can guess fairly accurately when the target has shot a bullet. In robocode, a robot can shoot bullets with powers ranging from 0.1 to 3, and this much energy is taken away from the robot's total energy pool. As the robot is capable of recording data between scans, the previous total energy can be compared against the current total energy to obtain the difference in energy. If the difference in energy is between 0.1 and 3, then it is safe to assume that the enemy has shot so the robot can move backwards or forwards to dodge the bullet. The practicality of this method is that it is a stop&go movement. This means that any linear targeting method will not work as efficiently on the robot because its initial velocity is 0 and only moves when a drop in energy is registered in the enemy. So the enemy is only able to shoot at the current position rather than predict it.

Another important detail is that some enemies are programmed to get close to you in order to keep track of you and shoot you constantly. This can be very devastating for the robot as it may end up trying to dodge bullets that are virtually impossible to dodge if there simply is not enough time to move out of the way. So what is done is place the function under a condition constraint where it will only trigger if and only if the distance between the enemy and the robot exceeds a threshold (in this case 142). Any less and the robot could end up wasting crucial bullet damage.

3.5 Shooting

3.5.1 Head-On

Head-on targeting is practically the simplest form of shooting there exists in robocode. It is simply a function which takes in three parameters: the robot's heading, the robot's gun heading and the enemy's bearing. By summing the robot's heading with the enemy's bearing, we obtain the absolute bearing of the enemy (because although robot heading is absolute, the enemy's bearing is relative to the robot's heading. Then subtracting the gun heading gives us the required angle to turn the gun and aim at the target.

3.5.2 Linear

Linear targeting requires more information and more calculations to predict the enemy's coordinates.

1. Consider the bullet speed depending on the power of the bullet using the provided formula in the robocode wiki.
2. Calculate the distance travelled by the bullet in 10 ticks is calculated. Call it *distanceBullet.*
3. Then calculate the impact time of our bullet with the enemy robot. Take a guess of 20 ticks approximately and find out the distance the enemy would travel at its velocity; *distanceEnemy.* Use an iteration until both *distanceBullet* and *distanceEnemy* are the same to obtain the time of collision.
4. Predict the enemy's coordinates at that time.
5. Measure the angle to turn the gun for to shoot at the predicted point.

3.5.3 Learning Robot

Learning agents are useful to improve and adjust the robot over rounds by recording and passing on variables which act as indicators of what it needs to modify (it would not be intelligent to keep doing something that does not work, instead of trying other methods). As described earlier, the robot has an integrated evaluation function to keep track of how efficient a gun mode is on the different enemies is encounters. By making variables static, the program is able to store data over rounds instead of getting reset to its original value. This implies that if one keeps record of the number of shots it has fired and the number of bullets it lands on the intended target (using the *onBulletHit* event), some form of history can be kept thus making the robot able to track which gun has been used on which enemy and which has landed more bullets.

4. EVALUATION METHODOLOGY

The robot is evaluated by observing the results at the end of the battles. Different fields of the robot will be weighed differently. Many of the components of the game can be adjusted to test the systems. A very useful feature of the robocode program is that you can specify how many rounds you want a battle to last. This facilitates evaluation greatly as one can simply establish a very large number of rounds for each battle instead of repeatedly running many battles with few rounds.

Another part of the game is the battlefield itself. The tests will be ran in a range of battlefields to put the robots in different events and most importantly, the enemies the hsck98 robot faces against will be predetermined to test for specific concepts of our systems.

4.1 Evaluating Minimum-Risk Movement

The strategy's main idea is to use enemies' positions to figure out the paths of least resistance, thus avoiding close combat. It is only natural that the marking system for this characteristic of the robot is the survival points attained at the end of the battles.

One key aspect of the game that affects the minimum-risk movement effectiveness is the size and number of enemies that the robot is facing against. The robot is likely to do worse in small battlefields where the empty spaces are lacking, therefore reducing the number of good options for the robot; and in crowded battlefields where the robot has no space to actually reach the desired destination because of other robots crossing paths. An indicator of how poor the upcoming melee battle is could be concluded using a simple ratio:

Number of enemies/ (battleFieldWidth*battleFieldHeight)

The higher this value, the worse the situation would be and the worse the robot would perform. In addition, a fatal factor of the survivability of the robot, in fact all robots, is the number of bullets received by crossfire. Although we minimize the chances by moving away from the targets thus not becoming their primary focus for firing, there is a higher chance of getting hit if the battlefield is small (at least if the value the indicator is high then there is a higher probability of the crossfire hitting an enemy).

4.2 Evaluating Dodging Skills

To test the function's effectiveness, 1v1 battles were set up and the total number of bullets shot by the enemy were counted and compared against the number of dodged bullets by the robot. By doing so, a percentage could be obtained. For easier evaluation the combat function was commented out during testing to concentrate the robot on dodging. The enemy chosen for this test was *TrackFire*. *TrackFire* shoots from a constant position at the enemy with a head-on targeting method. Another thing to notice is that the enemy robot's program was edited to print out the sum of the total number of shots it took and the total number of shots it landed at the end of the battle.

4.3 Evaluating Shooting

As explained earlier two different types of shooting modes are implemented: head-on and linear. In this case, both can be assessed using the same factor which is the bullet damage in 1v1 and the effectiveness of each one against different robots.

As a reminder, the bullet damage is meant to be used as a measurement of how well the firing systems did in a battle as an individual, not by comparing it to other robots. The reasoning behind this decision is because our movement strategy consists on getting away from robots so obviously it is very likely that the robot will end up isolated from others thus not firing as often. So bullet damage is not such a good evaluation factor for melee battles. On the other hand, the bullet damage can be very informative when it comes to 1v1's.

For head-on targeting, the best enemies to face it up against are any robots which stay still for long periods of time such as *Tracker* or *SittingDuck*. In terms of computations, the head-on targeting is obviously much cheaper than linear targeting as it requires fewer arguments and calls to functions in order to hit the target (although these might be negligible).

For linear targeting, the best enemies are those that tend to follow a straight line since the function extrapolates the enemy's coordinates believing it is going to keep moving in a straight line at a constant velocity. The best robot to observe its effectiveness is the robot from the sample package *Walls*. Walls goes around the walls aiming at the inside of the battlefield and shooting whenever it scans a robot. So when it is travelling from corner to corner, the linear targeting method is able to pinpoint its future coordinate and accurately land its shots.

Robots that present some challenge to this method include robots which include some form of stop&go strategy (eg: *MyFirstRobot*) or circular movement (eg: *SpinBot*). *SpinBot* is the enemy which our system performs worst on as it follows a circular pattern therefore disabling the robot from predicting its future position. Tests were also performed

5. EVALUATION RESULTS AND DISCUSSION

5.1 Evaluation Results Minimum-Risk Movement

As hypothesized earlier, the strategy implemented corresponds to lower survivability the smaller the size of the battlefield as seen in figure 2. The graph also shows another fact which is that the growth in average survival points achieved has a logarithmic growth against the size of the battlefield. This was to be expected as the closer the robot gets to the maximum number of survival points (in this case 80000 points), the more difficult it is to exceed the previous score. The reason for this is mainly because no matter how good the robot can dodge nearby enemies, there is no guarantee it will not be hit by crossfire.

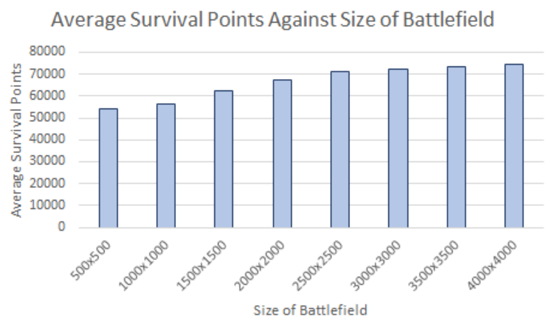

Figure 2: Average survival points against size of battlefield over 5 battles of 100 rounds each

5.2 Evaluation Results Shooting

Overall, there was an immense difference in bullet damage dealt between linear targeting and head-on targeting against *Walls*. This can be very apparent in figure 3 where the values from one to the other are always at least over 2000 damage points. The linear targeting showed excellence performance against linearly moving opponents which meant it could cover up for head-on's poor performance in this region. On the other hand, there were as expected a few enemies in the sample pool which the robot struggled to hit consistently and purposely. Amongst these, *SpinBot* was the worst adversary to match it up against. Both of the gun modes showed unsatisfactory results and the head-on targeting was actually more effective against it. The graph in figure 4 shows an average of 2491.8 bullet damage using linear and 2952.4 using head-on. However, not to be mistaken, the robot did not predict the circular motion of the *SpinBot* it just happened. In most cases, the enemy happened to be in the trajectory of the bullet that was aimed at its previous position or at its predicted position using the linear velocity from its previous point. Regardless of this fact, whichever executes the firing more adequately is used over as the primary gun mode.

5.3 Evaluating Dodging

The dodging skills were performed as explained previously and the result can be observed in figure 5. As seen, the average percentage of dodging is about 12.6%. It is a much lower success rate than anticipated and there are several reasons why this might be so. Starting with the fact that the robot is built extending the robot class and not the advancedRobot so it cannot multi-task. What this leads to is a lack of speed to update data on the enemy fast enough to react to the shots fired. Since it has to scan and then move, and it cannot scan to update until it is done moving. Hsck98 has an average scan rate of about 1 scan per 14 ticks which

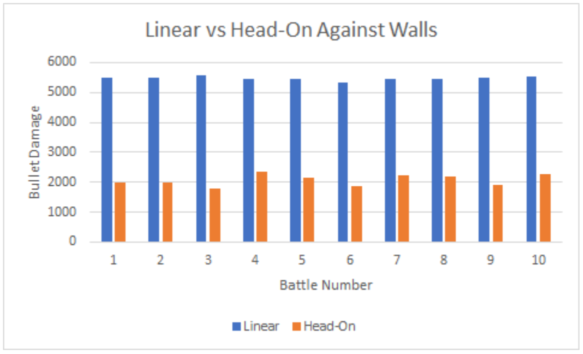

Figure 3: These are the results of facing hsck98 against Walls using either only linear targeting or only head-on targeting over 10 battles of 50 rounds each

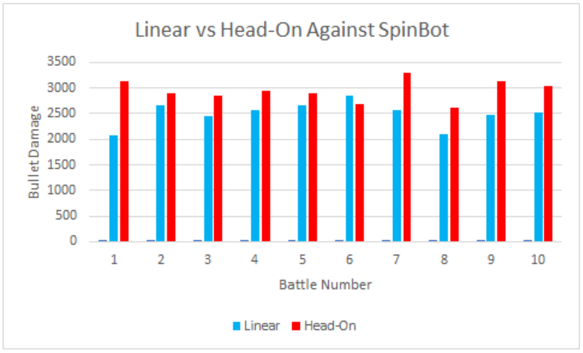

Figure 4: These are the results of facing hsck98 against SpinBot using either only linear targeting or only head-on targeting over 10 battles of 50 rounds each

is very slow considering the maximum number of shots is 1 per 15 ticks. Although it might be able to scan slightly faster than the shooting rate, it is all about timing. If the robot scans the enemy, at tick14 but the enemy shoots at like tick16, hsck98 will not notice this drop in energy until tick28 which is the next time it senses the enemy. By that time, the bullet will have travelled a considerable distance and in many cases, the bullet will be too near to dodge and because the robot is not designed to know how and when to readjust it own function timings, a cycle of mistakes will form.

This is a substantial disadvantage in the system however it is somewhat alleviated by the fact that the test was performed without the combat function. Although it might seem counter-intuitive, having the *fire()* function is beneficial to the robot. The robot is made so that it automatically does a 360 degrees scan after doing any action such as moving, firing, etc... This suggests that in the case the timing mentioned earlier has formed a never ending cycle, the *fire()* function will generate a different timing of scans thus aiding in updating the enemy's energy more often and breaking such loop.

Another reason why the function is kept in the program even after such poor results is because in cases where the robot faces against very offensive opponents, just a slight advantage such as a single bullet dodge can decide between victory or defeat.

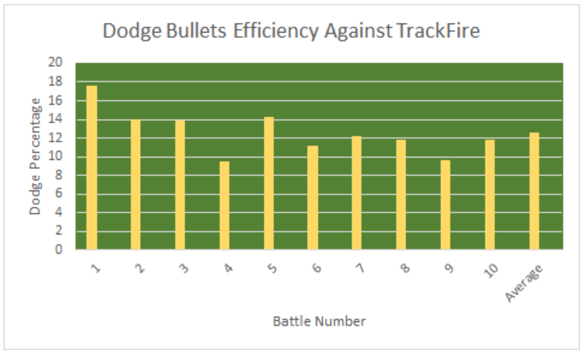

Figure 5: Percentages of the robot dodges against *TrackFire* over 10 battles of 50 rounds each

6. CONCLUSIONS

By the usage of AI techniques, the robot is able to carry out skillful movement melee movement mechanics by using a reflex agent with state. All the same, the 1v1 movement analyzed during dodging is not as effective but mostly because of the robot's physical limits, rather than the actual program. Shooting comes a bit short as it does not perform great against any robot with angular velocity of any sort. Nonetheless, its capability of learning which gun mode is suitable for each enemy is truly fascinating. Overall, it did relatively well against most sample robots but how well will it pair up against other CS students' robots? This relatively short yet intricate project has helped in learning about the extent of possibilities of AI. Although, simple AI techniques were used many tasks were accomplished and that goes to show what else anyone could do with more sophisticated AI. Another lesson learned is the basic procedure in creating a learning program and its implementation.

6.1 Possible Improvements and Ideas

This section will discuss the additional features that would be favourable to have. Although an attempt of the circular targeting was done in the process of creating the robot, this function would only be effective against enemies which move at a constant angular velocity. In many cases, robots have been designed to not follow constant velocities be it linear or angular, thus preventing prediction. So instead of doing either linear or circular, a pattern matching algorithm could implemented. This would cover both scenarios and much more since it would map the movement of the enemy to its last several movements to predict a pattern in its movements.

In order to build an almost impenetrable defense, the robot could have a bullet shielding function incorporated. Bullet shielding consists in predicting the gun heading of the enemy and calculating the linear trajectory of it to either dodge it or in the case, it is unavoidable a bullet is shot to intercept the enemy bullet's trajectory and thus countering its offensive plays. The idea would be great because just like minimum-risk movement saves energy by avoiding fights, bullet shielding could be used as a last resort to block the bullets and due to the physics of law in robocode, a bullet of 0.1 power could stop bullets of 3 thus saving even more energy.

Another add-on could be the implementation of the AdvancedRobot as this would ease the challenge of continuous scanning, movement and shooting simultaneously.

REFERENCES

[1] Minimum Risk Movement by Robowiki, `http://robowiki.net/wiki/Minimum_Risk_Movement`

[2] Linear Targeting by Robowiki, `http://robowiki.net/wiki/Linear_Targeting`

[3] Head-On Targeting by Robowiki, `http://robowiki.net/wiki/Head-On_Targeting`

[4] HawkOnFire general guide, `http://robowiki.net/wiki/HawkOnFire`

[5] Diamond general guide, `http://robowiki.net/wiki/Diamond`