**02 June -06 Jun** First off, I got the runtime down to a few minutes by replacing some function calls with look-up tables. It turns out that calling a function 7.5 million times before the policy converges eats up quite a few clock cycles.

I spent a lot of time this week working on finding a way to incorporate this pk framework into the MDP. One thing that seemed promising was to make the state definition be the agents target and then the current Pk’s for each target. This had the severe disadvantage of having multiple continuous state variables but had the advantage of incorporating the attrition rate into the state transition function. I believe this would give us a more faithful model. Since the attrition rate is time-varying, replanning would be required in parallel with execution of the policy on the actual vehicle, fortunately policy iteration makes replanning fast assuming that the system didn’t change too dramatically.

Before trying to implement the continuous state, I used the current implementation but replaced the reward function I was using. Previously the reward was the negative absolute value of the difference between current and desired totals. For this version I calculated the Pk on each target based on a nominal weapon effectiveness and attrition rate. I then made a linear combination of the difference between the state Pk and the desired Pk for each target, weighted by a priority. Interestingly, with this reward function the policy converges in many fewer iterations. For the scenario I use so that I can compare different methods, it converges in 86 iterations as opposed to 490.

The biggest problem with this method is that since each additional robot engaging a target has a smaller marginal effect on the Pk for the target than the previous one this reward function biases heavily towards a uniform distribution. This possibly could be accounted for by careful tuning of the priority weighting in the reward function though.

If we frame this problem as required Pk’s that we want to satisfy for each target or get close to if full satisfaction isn’t possible, that suggests decentralized constraint optimization which I looked at a few months ago and abandoned though I don’t recall why. I’ll spend the rest of the morning looking at this unless I remember why it was discarded.

On the subject of alternative methods, since you wanted me looking a little each week to see if any other techniques might be useful, yesterday I looked at backtracking search. The idea is to start with goal states and then undo all allowable actions. This gives a frontier of all states that are one move away from the goal. This is then repeated until every possible state has been generated, with only the first time a state is seen being kept. The action that was “undone” to generate a given state is the optimal action for that state as that action transitions to the state closes to the goal state.

What I discovered from doing this is obvious in hindsight that, assuming all of the other agents are static, there are *n* possible states for an agent to occupy at any given time where *n* is the number of targets. It might be sufficient for each agent to just pick the best of these *n* states and then reevaluate its state based on observed results of the actions of other agents.

For now, I just treated effectiveness and attrition rate the same for all agents and targets but I was careful not to do anything that would prevent switching this out for more accurate models later on.

Here is the histogram for the policy generated with the Pk reward function. In a proper policy, we’d expect targets 1 and 2 to appear in the policy more as they required the most agents to satisfy their assigned Pk.

