Chapter 1

Controllers

The SCOUt project is focused on creating a single intelligent controller that can be used across multiple environments to achieve multiple goals. This chapter covers how the intelligent controller's schema works and the heuristic controller schemas it is compared against. Heuristic controllers follow a set of logical steps to decide upon an action based on the given agent state. An individual heuristic controller must be created for goal and agent setup. The experiments in this project look at two different goals. The first is an anomaly searching goal to find a Human within an environment: FindHuman. The second is an element mapping goal to map out water presence in an environment: MapWater. A heuristic controllers are created for each of these goals. For a comparison base line, a random controller is also compared. The random controller will simply select a valid action at random until it is no longer operational, or it has completed the goal. All three controller types (random, heuristic and intelligent), are designed to operate within unknown environments using whatever sensors they have at hand.

1.1 Heuristic Controllers

FindHumanController and MapWaterController are the two different heuristic controllers designed for achieving their respective goals. Both use a similar action decision schema for completing an Operation. When the controller is called to choose an action, each valid action is given a score by analyzing values within the AgentState. The action with the highest score is then selected for the Agent to take. Valid scanning actions are all scored using the same function, scoreScanAction. Scoring valid movement actions involves several sub-functions tied to each element type in the Environment, but is used to score movement in the four cardinal directions. Each of the sub-functions within scoreMovmentAction calculate a sub-score for their corresponding element type using threshold analysis on the given QuadrantState corresponding to the movement direction. Once each sub-score has been returned to the scoreMovmentAction function, an

add function

add function

example sub-function

overall score is determined for moving in that direction. Heuristic controllers also keep track of each action taken during an Operation. This action history is used to discourage the controller from repetitive action selection.

The difference between the two heuristic controllers is found in their movement scoring functions. Scan actions are all scored using the same scoreScanAction function. The FindHumanController influences movement to cells that have higher decibel and temperature differentials as a Human Anomaly will likely be indicated by increased values in these element types. The MapWaterController focuses movement into quadrants that have fewer known WaterDepth values so it can move to these areas and scan for data. Both controllers' movement is also influenced by the presence of water and large elevation differentials. This will discourage the Agent from attempting moves that could result in damage.

1.2 Intelligent Controller

The SCOUtController operates using memory based reinforcement learning. After each operation, the SCOUt controller will store state-action pairs into its memory for later reference. A state-action pair (SAP) contains the action that the agent took, the state that the agent was in when it chose this action, and the short and long term rewards that the action received. In future operations, the SCOUt controller can search in its memory to find state-action pairs where the state is similar to the agent's current state. Once the controller has gathered SAPs with similar states, it will look at what actions were performed and the rewards that were given to predict the best action it should perform in its current state.

1.2.1 Memory

Memory is gathered from every operation that the SCOUtController is used in. When an operation has finished and long term rewards have been assigned to each action, the controller creates SAPs for every action that was chosen and selects a sub-set of them to be stored in memory. Saving only this sub-set cuts back on the size of the memory file and the computational time required while searching through memory for similar states. The current memory selectin method in this project will save the last 20 SAPs and then a uniform sampled sub-set of 5 percent of remaining SAPs from the operation. The last 20 actions are always saved because they typically are the most important events leading up to success or failure of the goal at hand. The remaining actions taken prior to these last 20 are uniformly sampled so that the memory will also contain information related to the agent's initial searching behavior. Each state-action pair is saved in the memory file as a Json object that can be decoded the next time the controller's memory is loaded.

1.2.2 State Comparisons

SAP memory is loaded into the controller before an operation begins. Once loaded, the states within the memory are all normalized to make variances within each state's data relative to all other states in the memory set. Normalization takes place for all numerical values stored within the AgentState. This includes:

- health
- energyLevel
- each ElementState's: percentKnownInSensorRange
- each QuadrantState's percentKnown averageValueDifferential immediateValueDifferential

Normalization follows this Guassian distribution equation suggested by **??**.

normalization eq

This makes data values that are more meaningful when studied by the agent. For example, if a controller was seeking out a human, it may look for increases in decibel values. In order for the controller to determine how much of an increase in decibel readings is accurate, it needs to have an idea of what is normal variation in decibel values versus a significant increase. Gaussian distribution provides this functionality through the calculated average and standard deviation within a data set. If the agent has gathered decibel readings in its north quadrant that are well above the standard deviation of readings it has stored in its memory, it should be considered significant.

Once the internal memory has been loaded and normalized, the controller can use the states of SAPs to compare against the agent's current state and predict the best action. Each time the controller is used to decide upon an action, it will compare its current state to states within its memory. State comparisons are calculated using the following algorithm: Comparison follows this algorithm:

style algo

1. Normalize the current state (how many STDs it falls outside of the average) 2. Calculate the difference for: a. health b. energyLevel c. elementStateDifferences = for each element state: i. hazardDifference = if (current == SAP) 1 else 0 ii. indicatorDifference = if (current == SAP) 1 else 0 iii. percentKnownInSensorRangeDifference = abs(SAP - current) iv. immediateValuesKnownDifference = abs(SAP - current) / 4 d. quadrantToQuadrantStateDifferences = for each current quadrant: i. quadrantStateDifferences = for each SAP quadrant: a. quadrantElementStateDifferences = for each element type: i. hazardDifference = if (current == SAP) 1 else 0 ii. indicatorDifference = if (current == SAP) 1 else 0 iii. percentKnownDifference = abs(SAP - current) iv. averageValueDifferentialDifference = if (current known SAP known) abs(SAP - current) else (if current known == if SAP knonw) 1 else 0) v. immediateValueDifferentialDifference = if (current known SAP known) abs(SAP - current) else (if current known == if SAP knonw) 1 else 0)

The state comparison algorithm generates difference calculations with movement and scanning action types in mind. The elementStateDifferences (item

c) are used when comparing the current state againts and SAP who's action was a scan action. For each elementStateDifference, the algorithm compares the similarity between how the elemet type is being studied (for hazard detection and goal related inication), as well as the percent of the element type known in range of the sensor and how many of the four adjacent cell's values are known. The hazard and indicator differences guide the controller to determine the importance and usage of the element types data. The percent known and immediate values known difference guide the controller to decide whether usage of an element type's sensor is efficient, or necessary. For example, if an agent does not have knowledge of the Elevation in adjacent cells, it couldn't confidently determine whether moving into one of those cells would result in taking fall damage. The quadrantStateDiffernces (item d) are used when the SAP's action was a movement action. Each quadrant for the current state is compared against every quadrant in the SAP's state. In doing so, this allows the controller to consider all orientations between the two states.

orientation diagram

Each orientation is important to consider because SOMETHING SOMETHING WORDS. Within each quadrant to quadrant comparison, the controller will consider the differences between values for each element type. These quadrantElementStateDifferences look at values related to the specific quadrant instead of the entire internal map as elementStateDifference compares.

Once these specific differences have been calculated, the controller must collapse them into a single, overall state difference to help predict the reward the agent will receive for each possible actions. To collapse the collection of state differences, the controller must apply different weights to each individual difference, and average the total of them all. There are two differnt calculations for overall state differences. The first is if the SAP's action was a scanning action, and the second is if is was a movement action. The equations for this weight-based difference calculation are as follows:

equationzzz

overallStateDifference(scanning) = (health * Whealth + energyLevel * Wenergylevel + overallElementStateDifferences * WelementStates) / 3

overallElementStateDifferences = sum(for each elementStateDifference: overallElementStateDifference * WelementState) / number of elementStateDifferences

overallElementStateDifference = (hazardDifference * Whazard + indicatorDifference * Windicator + percentKnownInSensorRangeDifference * WpercentKnownInSensorRange + immediateValuesKnownDifference * WimmediateValuesKnonw) / 4

overallStateDifference(movement) = (health * Whealth + energyLevel * Wenergylevel + overallQuadrantDifferences * Wquadrants) / 3

overallQuadrantDifferences = min(for each orientation: overallOrientationDifference)

overallOrientationDifference = (overallQuadrantDifference1 * Wquadrant + overallQuadrantDifference2 * Wquadrant + overallQuadrantDifference3 * Wquadrant + overallQuadrantDifference4 * Wquadrant) / 4

overallQuadrantDifference = sum(for each quadrantElementStateDifference:

overallQuadrantElementStateDifference * WquadrantState) / number of quadrantElementStateDifferences

overallQuadrantElementStateDifference = (hazardDifference * Whazard + indicatorDifference * Windicator + percentKnownDifference * WpercentKnown + averageValueDifferentialDifference * WaverageValue + immediateValueDifferentialDifference * WimmediateValue) / 5

1.2.3 Action Selection

Once an overall difference has been calculated between the agent's current state and states in memory, the controller can predict the reward that will be received for each possible action. An action's short and long term rewards are predicted from the averages of SAPs where: A) the considered action was selected, and B) the state difference from the current state is below a certain threshold. In addition to these predicted rewards, we also calculate a confidence value for the predictions. The lower the difference is between the current and SAP states, the higher the confidence will be. Additionally, the more SAPs that are considered in the prediction, the more confident the controller can be in the predicted reward.

confidence EQ

Bibliography