Surveillance Coordination and Operations Utility (SCOUt)

Keith August Cissell

June 2018

Abstract

asdf

Acknowledgement

Thanks to all my peeps

Contents

Chapter 1

Introduction

As research in the fields of autonomous systems and robotics have become more extensive, it is evident that there are a wide range of application for robots with integrated autonomy. There are rovers, drones and even aquatic robots that are capable of decision making in their own environments. The tasks that these robots carry out can greatly vary as well. This variance can cause a demand for distinct software and hardware to achieve each robot's given task. However, almost all autonomous robots operate similarly through their use of observation (typically with external sensors) and analytics of the data that is observed.

A great deal of research has been done in hybrid robots and creating hardware that is multifunctional to various tasks However, there is not an extensive amount of research on software with the capability to integrate with multiple robot compositions and tasks. Most of this is due to the fact that each robot has unique capabilities that do not overlap with many other robots. Autonomous robots seem to focus in on a certain niche and require their systems to be built from the ground up each time. This leaves the question of what pieces of autonomous control can be abstracted.

There are many evolutionary computing approaches that can be applied to decision making processes. These methods are commonly used in situations when there are a known number of controllable variables and a wide solution space to be explored. This makes them great candidates for creating a system which drives the decision-making process of autonomous robots. In particular, neural networks and deep neural networks trained in simulations seem to be a promising architecture for finding optimal control patterns in the diverse applications of autonomous robots.

This project approaches the problem from the bottom up. It looks at the very basics of autonomous robotics. This is: the collection of data from sensors, analytics of incoming data, and the output of response controls. Additionally, these three steps are repetitively being performed to achieve a given objective. I have broken this project into three phases. The first phase involves setting up a simulation environment to be used for training the autonomous system. Next, a graphical interphase will be integrated with the simulation data to allow for

easy debugging. Finally, an Artificially Intelligent system will be trained to take in various sets of environmental data as inputs, make decisions based on these inputs and its current objective, and produce a response.

The project is still a work in progress and this paper will only present phases one and two. These phases cover the procedural environment generator and the graphical interface that pairs with it. The implementation of the abstracted autonomous system will come in future work. The main topic covered looks at the representation and formulaic production of data that will be used to represent an environment. The graphical interfaces capabilities will also be touched on. For the AI component, we will look at the various evolutionary methods that hold great potential for our given problem setup.

Chapter 2

Related Work

Related works

Chapter 3

Methodology

The project's simulation consists of three main components: environments, agents and the interactions between the two. A wide range of diversity is required for both the environments and agents represented in simulation.

3.1 Environment Build Tool

Representing any environment is a tricky process. A simulation needs to balance simplicity and coverage when modeling an environment. Leave out too much from the model and it won't reflect real word scenarios. Trying to model too much can consume time and effort that could instead be used to run real world experiments. The SCOUt environment build tool captures important details that are necessary for agent-environment interactions to be simulated, while remaining simple to implement and understand. The tool is also highly abstracted so that more details can easily be added as needed while still maintaining a defined build process.

Build Process

1. User Fills out dynamically generated template. (separate diagram?)
2. Builder initializes a grid of empty cells
3. Element seeds are used to populate each present element type into the gird of cells
4. Terrain modifications are applied to manipulate their related element(s)
5. Anomalies are placed randomly within the environment
6. Anomaly effect(s) are applied to corresponding element(s) in neighboring cells

The environment build tool provides a Graphical User Interface (GUI) for creating and visualizing environments. Electron is used to simulate a web page

Possibly cite Electron

contained within a standalone desktop application. This allows the front end to be written in JavaScript, HTML and CSS and handle communication to the back end via http over a localhost network. Scala library http4s is used to create a server on a localhost network for handling the http requests from the front end. This architecture allows all data creation and manipulation to be isolated in Scala on the back end, while allowing user interactions with the data to take place on the front end. Launching the app starts up the Scala server in a new terminal and opens the Electron window which will begin attempts to establish communication with the server.

Possibly cite http4s

Once connection between the server and GUI have been established the user can choose to generate a random environment or build a custom environment. For a random environment, the user only inputs the name and size of the environment and all other variables are selected by the server. Building a custom environment steps the user through a series of form pages to create an environment template.

Load environment or template from file

Environment generation is guided by an environment template that holds the necessary data to build a specific environment. Use of a dynamic template to allow representation of a wide range of environments while allowing re-creation of multiple random environments based on the same template. The goal of each template is to provide influence over the values generated so that a specific environment can be modeled with minimal input. To-do ...

environment template table

Each **Environment** created is represented as an n x m 2D grid of uniformly sized s x s square **Cell**, whose size is specified by a scaling factor. Along with positional data, each **Cell** contains information about the different elements and anomalies present within their s x s area. An **Element** is a generalized object that represents one specific environmental attribute such as the elevation or temperature. An **Anomaly** represents some object present within the cell that could be of interest. Anomalies often have an effect on element values in their surrounding area which makes them "traceable".

Element and Anomaly are abstract data structures that each provide a basic object that is then extended to create specific instances. They all share core members that allow handling to be generalized when being manipulated or interacted with. Each instance also has an associated seeding process to automate their population within the environment. Seeding processes are specific to each instance of an Element or Anomaly.

Element seeds require input variables to be provided in order to drive their population process within the environment.

3.2 Agent Representation

Agents within this experiment have a core set of attributes and abilities, along with a set of sensors and a controller. The core attributes for an agent are health, energy level, a system clock, an internal map and its current position. Because SCOUt is focused on purely observational interactions with its environment, an agent only has two categories of actions that can be taken: movement and

scanning. The agent can attempt to move one cell at a time in any of the cardinal directions. This allows the agent to re-assess after each movement attempt. Scanning collects information about the agent's immediate environment and updates internal map. The list of scan actions that an agent can perform is based on the set of sensors the agent is equipped with. The controller is in charge of analyzing the current state of the agent and deciding the next action to be performed. This project focuses on creating a single controller (SCOUt) that highly adaptable to wide ranges of agent configurations, environments and operations.

3.2.1 Movement

3.2.2 Sensors

3.2.3 State Representation

3.2.4 Controllers

In this experiment three different controllers are created and compared for their performance.

3.3 Actions and Rewards

The SCOUt agent uses a

Chapter 4

Experimentation

Trial setups and data interpretation

Chapter 5

Results

What did the results yield and what can we infer from this

Chapter 6

Conclusion

Conclude stuff