An Introduction to Robotics and Artificial Intelligence

Sahit Chintalapudi

hack.gt

gtagency.github.io

Outline

- Introduction to the robots
- Velocity Control
- Gradient Descent
- Position Control
- Maze Solving
- (Brief Intro to) Multi-Agent Systems
- Robot Homing

The Robotarium

“If you have a good[sic] idea, you should be able to run your idea and test it on real robots”

Getting Started

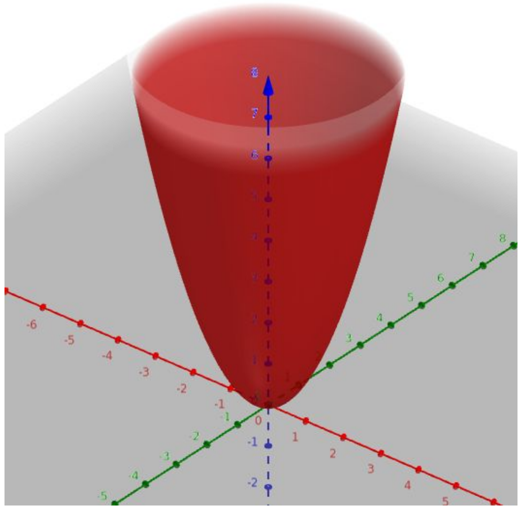
- Install Python3
  - https://www.python.org/downloads/
- Install git
  - https://git-scm.com/downloads
- Setup the Robotarium Simulator (Mac/Linux)
  - `git clone https://github.com/robotarium/robotarium_python_simulator`
  - `cd robotarium_python_simulator`
  - `git checkout gritsbotx`
  - `sudo pip3 install .`
- Setup the workshop starter code (Windows - In a command prompt as administrator)
  - `git clone https://github.com/robotarium/robotarium_python_simulator`
  - `cd robotarium_python_simulator`
  - `git checkout gritsbotx`
  - `pip install .` ON WINDOWS TRY `py -3 -m pip install .`
- Setup the workshop starter code
  - `git clone https://github.com/chsahit/hgt_workshop`

Fundamentals: Velocity Control

- Let’s get familiar with the robotarium library and simulator
- Question: Given a single robot, how we can set its velocity?
- Answer: Using a text editor of your choice, open set_velocity.py

Exercise 1: Setup

- Our robot is sitting somewhere on the side of a bowl and we want it to navigate to the bottom of the bowl
- We can model the bowl as $f(x) = x_1^2 + x_2^2$ and its' *gradient* as $\nabla f = [2x_1, 2x_2]$
  - $f: R^n \rightarrow R$



Exercise 1: Gradient Descent

- Gradient Descent is an algorithm that looks for (local) minima of a function f
- Start out with a random vector x_0
- The update rule we use is

$$x_{t+1} = x_t - \eta \nabla f(x_t)$$

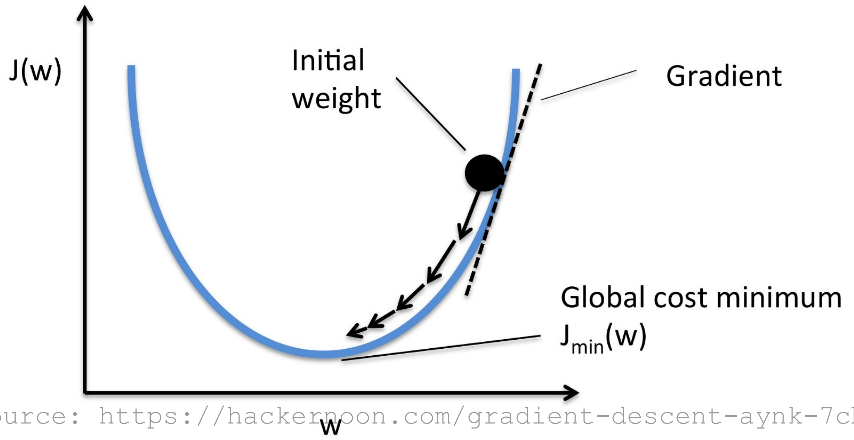


Image source: https://hackernoon.com/gradient-descent-aynk-7cbe95a778da

Exercise 1 Continued: Momentum

- Let’s look at functions with multiple minima
- What problem do we run into here? How can we address it?

Exercise 1 Continued: Momentum

- The robot gets stuck in a “local minima”
    - All the gradients are 0 but we can clearly reach lower values
- New update rule. Let x_0 be initialized to a random vector and initialize v to 0

$$v_{t+1} = \mu v_t - \eta \nabla f(x_t)$$

$$x_{t+1} = x_t + v_{t+1}$$

Fundamentals: Position Control

- Define the pose of the robot as the vector [x, y]
  - This is as opposed to the velocity based control we were previously using, which defined the pose as [$\dot{x}$, $\dot{y}$]
- Open up set_pose.py in an editor of your choice so we can see how to do this

Fundamentals: Search Graphs

- A graph is an object G = (V, E) where V is a set of *vertices* and E is a set of *edges* connecting those vertices
  - A set's elements are *unique*; there are no repeat values!
- The vertex set will be defined by the states the robot can take on
- The edge set will be defined by the actions the robot can take at every state to transition between states
- Each edge has an associated *cost*, in this example the cost is the length of the edge.

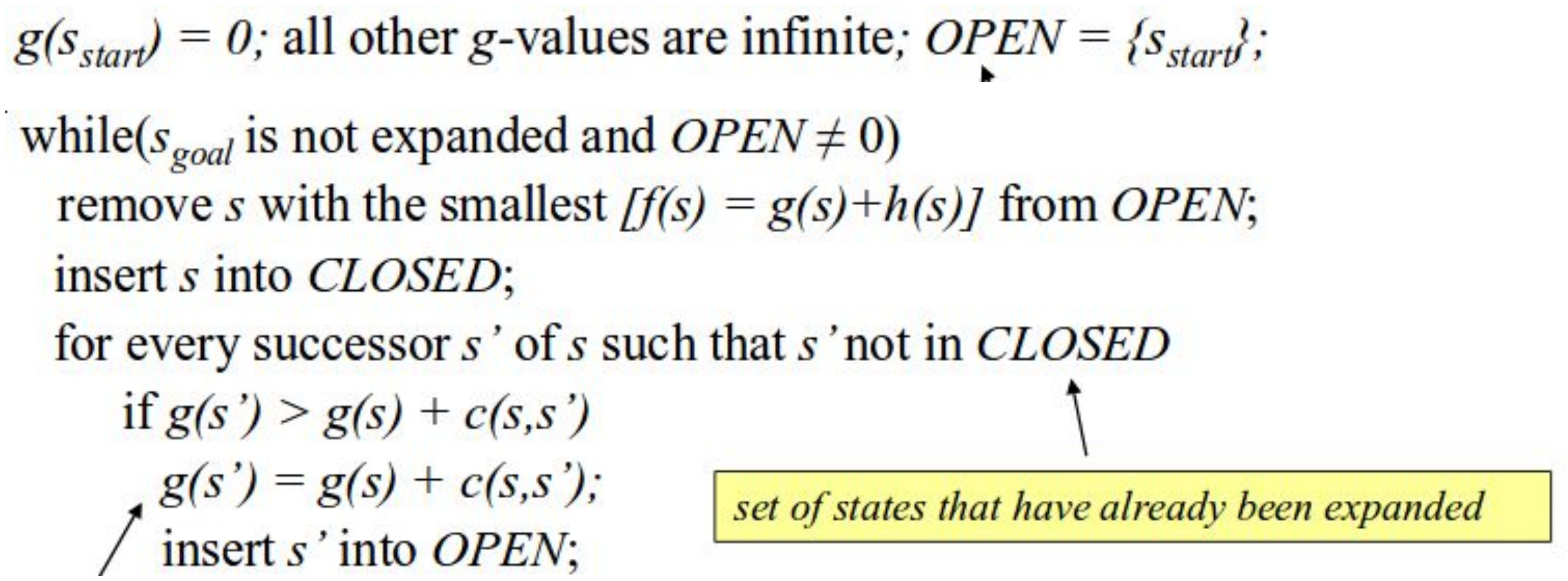
Fundamentals: Graph Searches

- Given a graph, start vertex, and end vertex, we want to find a list of edges that connect the start and end vertices. This ordered list of edges is called a *path*.
- The cost of a path is the sum of the edge costs. We want to minimize this cost.
- A *priority queue* is a data structure that stores (priority, data) tuples.
- Popping from a priority queue returns the (in this case) lowest priority data

Fundamentals: Heuristics

- A function h(s) is a *heuristic* function that maps a state to its' "heuristic value".
  - The value is *admissible*: the heuristic is never greater than the true cost between s and the goal
  - The value is consistent: given a state s and an arbitrary neighbor n, $h(s) \leq c(s, n) + h(p)$
- Why is the euclidean distance between two states considered a valid heuristic?

Fundamentals: A*



```
$g(s_{start}) = 0$; all other $g$-values are infinite; $OPEN = \{s_{start}\}$;
while($s_{goal}$ is not expanded and $OPEN \neq 0$)
    remove $s$ with the smallest $[f(s) = g(s)+h(s)]$ from $OPEN$;
    insert $s$ into $CLOSED$;
    for every successor $s'$ of $s$ such that $s'$ not in $CLOSED$
        if $g(s') > g(s) + c(s,s')$
            $g(s') = g(s) + c(s,s')$;
            insert $s'$ into $OPEN$;
```

set of states that have already been expanded

Source: Maxim Likhachev http://www.cs.cmu.edu/~maxim/classes/robotplanning_grad/lectures/astar_16782_fall18.pdf

Exercise: Solve a maze

- The robot has a green food pellet in the environment, but there are a lot of obstacles in the way!
- We can use A* to navigate the robot to the food pellet quickly

Fundamentals: Multi-Agent Systems

- A robot is considered an autonomous *agent*
- When we have multiple robots operating together in an environment, it is called a multi-agent system
- Examples:
    - Autonomous vehicles that can communicate driving down a road
    - A swarm of drones scattering through an environment for monitoring purposes
    - Modeling schools of animals (e.g fish or ants)

Exercise: Robot Homing

- Let’s add another agent to the mix at the robotarium and see what happens!
- We have one robot that is randomly moving around the environment
- Program a second robot to “hone in” on the first