Stack Cheese

ECE578

Brass, Brooks, Hull, Mayers, Minko

Electrical and Computer Engineering
Portland State University
2019-11-05

Contents

1 Overview

As a new design scheme, we changed the *Policemen and Thief* to the less violent *Mice and Cheese*.

Design a game with 3 Viking Bots: two dressed as mice and the third as a wedge of cheese. The goal of the game is to keep the cheese away from the hungry maws of the two little mice. The game uses object recognition to find the location of each game piece on the board. The program builds a game board and determines the proper strategy so that the agents can move effectively in the right direction. The user goes first and can control the course of the cheese, and the mice will continuously move towards the cheese after each turn. Will the cheese escape the mice? Or will it get eaten? Tune in and find out next time on Perkowskis comedy cartoon hour.

2 Hardware

The hardware for our project consists of 3 robots and all their associated parts. In the past, this project utilized 2 "Viking Bots" and a more massive hexapod robot. Our requirements for the project were to replace the hexapod robot with another Viking Bot and to improve the robustness of the hardware. This would ensure that our hardware can run more consistently, at a faster pace, and in more diverse conditions than were possible in previous implementations of the project.

2.1 Goals

1. Purchase and assemble a new Viking Bot
2. Assess what hardware was left over from previous implementations of project
3. Upgrade and standardize battery packs for all robots
4. Improve wiring reliability and cable management
5. Ensure that the robots can run consistently at top speed for fast gameplay endenumerate

2.2 Design

We assembled our projects robot cars from an inexpensive kit, which was quick to assemble. The Viking Bot robot kit consists of the following parts:

- An acrylic sheet with mounting holes and cutouts for wires
- A set of 2 DC motors and wheels
- A swivel caster for a back wheel
- A battery pack (AA battery size)
- An L298N H-Bridge Module
- A Raspberry Pi microprocessor board

Due to the nature of using a "kit" robot, we didnt have a lot of latitude to design the hardware we were using. However, we undertook the development of a better battery and power management system for the robots after discovering the following problems:

- Batteries we inherited had differing voltages, causing robots to function differently from one another
- Some robots used multiple battery packs to achieve uniform voltages, adding extra weight to the robot
- The VIN port powered the Raspberry Pi's, causing damage to the boards.
- H-Bridge modules were providing inconsistent outputs and current limiting the Raspberry Pi's

3 Movement

Each Viking Bot has a raspberry pi that sends signals to an L298N H-Bridge. We created the controls using simple python functions that send commands which control the direction, power, and speed of the motors. It is possible to sign into the board remotely and send instructions using the wifi capabilities of the Raspberry Pi.

3.1 Goals

Control each robot remotely Control speed, timing, and direction of robots Streamline all hardware to allow identical control schemes for each robot

3.2 Design

The Viking Bots are functionally Braitenberg vehicles. The control is simple and sent to the H-Bridge via the Raspberry Pi. We control the movement of the robots by the applied power and the direction of the motors rotation. The robots are fundamentally simple but need a mechanism to control them remotely.

3.3 Implementation

The Viking Bots are feedback-driven agents that we control through the overall game program. The user gives the program instructions in which way they want the bots to move. The program interprets this command and decides the turn direction and motor-driven distance.

3.4 Challenges

The lack of documentation regarding movement is a considerable challenge. The primary control program on each robot is simple and easy to follow; however, the mechanics of the game and functionality have been the biggest hurdle.

- No documentation
- No instructions for game use
- No instructions on setup
- No comments in code

3.5 Planned Improvements

- Better documentation
- Example or walkthrough of how to set up a game
- Inline code comments explaining how each function operates

4 Some sample commands that shouldn't be in the final document

Math me

$$\begin{aligned} c_{max} &= 5.930\,85\,\mathrm{V} - 5\,\mathrm{V} \\ c_{max} &= 0.930\,85\,\mathrm{V} \end{aligned} \tag{1}$$

$$\begin{aligned} c_{final} &= 5.5\,\mathrm{V} - 5\,\mathrm{V} \\ c_{final} &= 0.5\,\mathrm{V} \end{aligned} \tag{2}$$

5 Another Section

5.1 Diagram

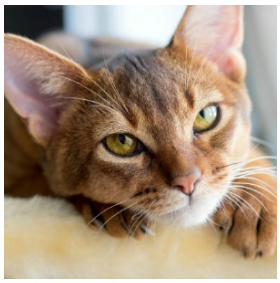

Figure 1: nyaaaaan

5.2 Analysis

Header 1	Header 2
Cell 1	Cell 2

Table 1: A Sweet Table

6 Q & A

1. *How does it work?*
   Black magic

7 MATLAB Code of Mathematical Analysis

```
% Robot cop/thief model
% Kai Brooks
% github.com/kaibrooks
% 2019
%
% Does a thing
%

% 3 axes of motion
% action space for agent:
% stay, left, right, up/down
% action space = 4
%
% state actions for agent:
% independent
% dependent on 1 robber
% dependent on 2 robber
% dependent on both
%
% total state space = 16


% Init
clc
close all
clear all
format
rng('shuffle')


% generate initial chromosome

lengthX = 6;
lengthY = 6;
maxPop = 10;

% generated internal vars
board = zeros(lengthX);
chromLength = lengthX * lengthY * 3; % size of the board,
    *3 for 3 bits

% generate chromosome
for n = 1:maxPop
    population(n,:) = round(rand(1,chromLength));
end
```

```
45
46
47
48
49  % mix
50
51  board(3,1) = 1; % starting position
52  lastPosY = 3;
53  lastPosX = 1;
54
55  % get nearby spaces
56  nextY = [lastPosY-1 lastPosY+1];
57  nextX = [lastPosX-1 lastPosX+1];
58
59  xw5x
60  % zero moves to large (off the board)
61  nextY(nextY>=lengthY) = 0;
62  nextY(nextY<=1) = 0;
63
64  % zero moves too small (off the board)
65  nextX(nextX>=lengthX) = 0;
66  nextX(nextX<=1) = 0;
67
68
69  % evaluate population
70  for i = 1:maxPop
71
72
73
74  end % 1:maxPop
75
76
77  % rebreed
78
79  % display
```

8 Python Code for A Thing

```
1  from camera_system import Camera
2  from object_detector import Detector
3  from strategy import Strategy
4  from graph_builder import GraphBuilder
5  from control_system import Controller
6  import logging
7  import sys
8  import time
9  import json
10 import random
```

```
11
12  WEIGHT_PATH = '../model/custom_tiny_yolov3.weights'
13  NETWORK_CONFIG_PATH = '../cfg/custom-tiny.cfg'
14  OBJECT_CONFIG_PATH = '../cfg/custom.data'
15  ROBOTS_CONFIG_PATH = '../cfg/robots.json'
16
17  logger = logging.getLogger(__name__)
18
19
20  class FakeGame:
21      def __init__(self):
22          self.camera = Camera(None, draw=False)
23          self.display_camera = Camera(None, window_name='
                labeled')
24          centers = []
25          with open('centers.txt', encoding='utf-8', mode='
                r') as file:
26              for line in file:
27                  center = tuple(map(float, line.strip().
                        split(' ')))
28                  centers.append(center)
29          self.centers = centers
30          self.graph_builder = GraphBuilder(self.centers)
31          self.orders = ['thief', 'policeman1', 'policeman2
                ']
32          self.strategy = Strategy(self.orders)
33          self.object_list = {
34              "thief": {
35                  "confidence": 0.99,
36                  "center": self.centers[6],  # (width,
                        height)
37                  "size": (0.15, 0.10),  # (width,height)
38              },
39              "policeman1": {
40                  "confidence": 0.99,
41                  "center": self.centers[1],  # (width,
                        height)
42                  "size": (0.15, 0.05),  # (width,height)
43              },
44              "policeman2": {
45                  "confidence": 0.99,
46                  "center": self.centers[3],  # (width,
                        height)
47                  "size": (0.15, 0.05),  # (width,height)
48              }
49          }
50          self.counter = 0
51          self.thief_movements = [13, 14, 15, 16]
52          self.escape_nodes = {10}
```

```
53          self.graph = None
54          self.objects_on_graph = None
55          self.instructions = None
56
57      def forward(self):
58
59          image = self.camera.get_fake_gaming_board()
60          self.display_camera.draw_boxes(image, self.
                object_list)
61          self.display_camera.display(image)
62
63          # build a graph based on object list
64          graph, objects_on_graph = self.graph_builder.
                build(self.object_list)
65
66          self.graph = graph
67          self.objects_on_graph = objects_on_graph
68
69          # generate instructions based on the graph
70          instructions = self.strategy.
                get_next_steps_shortest_path(graph,
                objects_on_graph)
71          logger.info('instructions:{}'.format(instructions
                ))
72
73          # instructions['thief'] = [objects_on_graph['
                thief'], self.thief_movements[self.counter]]
74          self.instructions = instructions
75
76          self.counter += 1
77          for key, value in instructions.items():
78              self.object_list[key]['center'] = self.
                    centers[value[1] - 1]
79          time.sleep(1)
80
81          image = self.camera.get_fake_gaming_board()
82          self.display_camera.draw_boxes(image, self.
                object_list)
83          self.display_camera.display(image)
84
85      def is_over(self):
86          """
87          Check if the game is over.
88
89          Returns
90          -------
91          game_over: bool
92              True if the thief is at the escape point or
                    the policemen have caught the thief,
```

```
                otherwise False.
93          """
94          game_over = False
95          if self.instructions is None or self.
                objects_on_graph is None or self.graph is None
                :
96              return game_over
97          if 'thief' in self.objects_on_graph:
98              if self.objects_on_graph['thief'] in self.
                    escape_nodes:
99                  game_over = True
100                 logger.info('The thief wins!')
101             else:
102                 for name, instruction in self.
                        instructions.items():
103                     if name != 'thief':
104                         if self.instructions['thief'][1]
                                == instruction[1]:
105                             game_over = True
106                             logger.info('The policemen
                                    win!')
107         return game_over
108
109     def get_report(self):
110         """
111         Generate a game report(json, xml or plain text).
112
113         Returns
114         -------
115         game_report: object or str
116             a detailed record of the game
117         """
118         game_report = None
119         return game_report
120
121     def shuffle(self):
122         random.randint(5, 10)
123
124
125 class Game:
126     """
127     Each game is an instance of class Game.
128     """
129
130     def __init__(self, weight_path, network_config_path,
            object_config_path, robots_config_path):
131         """
132         Load necessary modules and files.
133
```

```
134         Parameters
135         ----------
136         weight_path: str
137             file path of YOLOv3 network weights
138         network_config_path: str
139             file path of YOLOv3 network configurations
140         object_config_path: str
141             file path of object information in YOLOv3
                network
142         robots_config_path: str
143             file path of robots' remote server
                configuration
144         """
145
146         # fix robot movement order
147         self.orders = ['thief', 'policeman1']
148         # self.orders = ['policeman1', 'policeman2']
149         # self.orders = ['thief', 'policeman1', '
               policeman2']
150
151         # initialize internal states
152         self.graph = None
153         self.objects_on_graph = None
154         self.instructions = None
155
156         # set up escape nodes
157         self.escape_nodes = set()
158
159         # construct the camera system
160         self.camera = Camera(1)
161
162         # construct the object detector
163         self.detector = Detector(weight_path,
               network_config_path, object_config_path)
164
165         # load gaming board image and get centers'
               coordinates of triangles
166         self.gaming_board_image = self.camera.get_image()
167         self.centers = self.detector.detect_gaming_board(
               self.gaming_board_image)
168
169         # construct the graph builder
170         self.graph_builder = GraphBuilder(self.centers)
171
172         # construct the strategy module
173         self.strategy = Strategy(self.orders)
174
175         # construct the control system
176         self.controller = Controller(self.detector, self.
```

```
            camera.get_image, robots_config_path)
177
178         # connect to each robot
179         self.controller.connect()
180
181     def is_over(self):
182         """
183         Check if the game is over.
184
185         Returns
186         -------
187         game_over: bool
188             True if the thief is at the escape point or
                the policemen have caught the thief,
                otherwise False.
189         """
190         game_over = False
191         if self.instructions is None or self.
                objects_on_graph is None or self.graph is None
                :
192             return game_over
193         if 'thief' in self.objects_on_graph:
194             if self.objects_on_graph['thief'] in self.
                    escape_nodes:
195                 game_over = True
196                 logger.info('The thief wins!')
197             else:
198                 for name, instruction in self.
                        instructions.items():
199                     if name != 'thief':
200                         if self.instructions['thief'][1]
                                == instruction[1]:
201                             game_over = True
202                             logger.info('The policemen
                                    win!')
203         return game_over
204
205     def shuffle(self):
206         random.randint(5, 10)
207
208     def forward(self):
209         """
210         Push the game to the next step.
211         """
212         # get objects' coordinates and categories
213         image = self.camera.get_image()
214         object_list = self.detector.detect_objects(image)
215
216         # build a graph based on object list
```

```
217             graph, objects_on_graph = self.graph_builder.
                    build(object_list)
218             self.graph = graph
219             self.objects_on_graph = objects_on_graph
220
221             # generate instructions based on the graph
222             instructions = self.strategy.
                    get_next_steps_shortest_path(graph,
                    objects_on_graph)
223             self.instructions = instructions
224             logger.info('instructions:{}'.format(instructions
                    ))
225
226             if self.is_over():
227                 return
228             # move robots until they reach the right
                    positions
229             while not self.controller.is_finished(self.
                    centers, object_list, instructions):
230                 # obtain feedback from camera
231                 image = self.camera.get_image()
232                 object_list = self.detector.detect_objects(
                        image)
233
234                 # calculate control signals
235                 control_signals = self.controller.
                        calculate_control_signals(
236                     self.centers, object_list, instructions)
237
238                 # cut extra signals
239                 real_signals = []
240                 for name in self.orders:
241                     for signal in control_signals:
242                         if signal['name'] == name:
243                             # if True:
244                             real_signals.append(signal)
245                     if len(real_signals) > 0:
246                         break
247
248                 # update internal states
249                 self.controller.update_state(object_list)
250
251                 # move robots
252                 self.controller.move_robots(real_signals)
253
254                 # obtain feedback from camera
255                 image = self.camera.get_image()
256                 object_list = self.detector.detect_objects(
                        image)
```

```
257
258             # update internal states
259             self.controller.update_state(object_list)
260
261     def get_report(self):
262         """
263         Generate a game report(json, xml or plain text).
264
265         Returns
266         -------
267         game_report: object or str
268             a detailed record of the game
269         """
270         game_report = None
271         return game_report
272
273
274 def main():
275     # set up logger level
276     logger.setLevel(logging.DEBUG)
277     handler = logging.StreamHandler(sys.stdout)
278     handler.setLevel(logging.DEBUG)
279     logger.addHandler(handler)
280
281     # parse config file
282     if len(sys.argv) > 1:
283         config_path = sys.argv[1]
284     else:
285         config_path = '../cfg/game_config.json'
286     with open(config_path, encoding='utf-8', mode='r') as
        file:
287         config = json.load(file)
288
289     # load game parameters
290     weight_path = config['weight_path']
291     network_config_path = config['network_config_path']
292     object_config_path = config['object_config_path']
293     robots_config_path = config['robots_config_path']
294
295     # construct a game logic
296     game = Game(weight_path, network_config_path,
        object_config_path, robots_config_path)
297     # game = FakeGame()
298     # start the game logic
299     while True:
300         input('Press ENTER to the start a game:')
301
302         # keep running until game is over
303         while not game.is_over():
```

```
304                 game.forward()
305
306             # get the game report
307             report = game.get_report()
308
309             # display the game report
310             print(report)
311
312             # shuffle the robots on the gaming board
313             # TODO: finish shuffle() function
314             game.shuffle()
315
316
317 if __name__ == '__main__':
318     main()
```