ME 210: Intro to Mechatronics
Lab 0 - Programming Raptors on a Sparki

Koichi Kimoto

1. Q2.5 - Why doesn't the LED blink at 0.5Hz?

Because in these functions:

```c
1  void handleMoveForward(void) {
2    raptor.LeftMtrSpeed(HALF_SPEED);
3    raptor.RightMtrSpeed(HALF_SPEED);
4    delay(MOTOR_TIME_INTERVAL);
5    state = STATE_MOVE_BACKWARD;
6  }
7
8  void handleMoveBackward(void) {
9    raptor.LeftMtrSpeed(-1*HALF_SPEED);
10   raptor.RightMtrSpeed(-1*HALF_SPEED);
11   delay(MOTOR_TIME_INTERVAL);
12   state = STATE_MOVE_FORWARD;
13 }
```

we can see in lines 4 and 11, the

```c
delay(MOTOR_TIME_INTERVAL);
```

is blocking the checking of the timer expiration in the function `checkGlobalEvents();` in the loop:

```c
1  void loop() {
2    checkGlobalEvents();
3    switch (state) {
4      case STATE_MOVE_FORWARD:
5        handleMoveForward();
6        break;
7      case STATE_MOVE_BACKWARD:
8        handleMoveBackward();
9        break;
10     default: // Should never get into an unhandled state
11       Serial.println("What is this I do not even...");
12   }
13 }
```

(1) Timer expiration is checked here

(2) Both states contain blocking code

2. Q2.6 - Identifying Thresholds

Thresholds:

```c
LIGHT_THRESHOLD = 75;
LINE_THRESHOLD = 350;
```

Code to check the light and line values (which was used to determine the thresholds):

```c
void check_light_thres(void) {
  uint16_t currLightLvl = raptor.LightLevel();
  Serial.print("Current Light Level: ");
  Serial.print(currLightLvl);
  Serial.println(".");
}

void check_line_thres(void) {
  Serial.println("Current Line levels: ");

  Serial.print("Line Left: ");
  Serial.print(raptor.LineLeft());
  Serial.println(".");

  Serial.print("Edge Left: ");
  Serial.print(raptor.EdgeLeft());
  Serial.println(".");

  Serial.print("Line Center: ");
  Serial.print(raptor.LineCenter());
  Serial.println(".");

  Serial.print("Edge Right: ");
  Serial.print(raptor.EdgeRight());
  Serial.println(".");

  Serial.print("Line Right: ");
  Serial.print(raptor.LineRight());
  Serial.println(".");
}
```

3. Q2.7 Helper Functions for Raptor

```c
uint8_t TestForLightOn(void) {
  if (raptor.LightLevel() >= LIGHT_THRESHOLD) return 1;
  else return 0;
}

void RespToLightOn(void) {
  state = ATTACK;
  raptor.LeftMtrSpeed(FULL_SPEED);
  raptor.RightMtrSpeed(FULL_SPEED);
}

uint8_t TestForLightOff(void) {
  if (raptor.LightLevel() < LIGHT_THRESHOLD) return 1;
  else return 0;
}

void RespToLightOff(void) {
  state = LURK;
  raptor.LeftMtrSpeed(0);
  raptor.RightMtrSpeed(0);
}

uint8_t TestForFence(void) {
  if (raptor.ReadTriggers(LINE_THRESHOLD)) return 1;
  return 0;
}

void RespToFence(void) {
  state = BACK;
  backingTimer.reset();
  backing = true;
  raptor.LeftMtrSpeed(-1*HALF_SPEED); // reverse at an angle
  raptor.RightMtrSpeed(-75);
}
```

4. Q2.8 Simple Test Programs

```c
// turn motors on
void motor_on(void){
  raptor.LeftMtrSpeed(100);
  raptor.RightMtrSpeed(100);
}
```

```
6
7   // turn motors off
8   void motor_off(void) {
9     raptor.LeftMtrSpeed(0);
10    raptor.RightMtrSpeed(0);
11  }
12
13  // get char from keyboard
14  char get_char(void) {
15    return Serial.read();
16  }
17
18  // prt value to serial monitor
19  void prt_value(void) {
20    Serial.println("Hello World");
21  }
22
23  // read light sensor
24  uint16_t read_light_lvl(void) {
25    return raptor.LightLevel();
26  }
27
28  // beep buzzer
29  void Beep(void) {
30    raptor.Beep(100, 5000);
31  }
32
33  // turn on RGB LED
34  void turn_on_RGB(void) {
35    raptor.RGB(RGB_WHITE);
36  }
37
38  // Test IR sensors
39  bool test_IR(void) {
40    uint8_t triggerState = raptor.ReadTriggers(LINE_THRESHOLD);
41    if (triggerState) return true;
42    else return false;
43  }
```

4. Q3 Design & Implementation

My design mainly relies on the raptor having four states:

- **Lurk:**
  - The raptor is in this state if the light detector is below the threshold.
  - The raptor will turn off the motors when entering this state.
- **Attack:**
  - The raptor is in this state if the light detector is above the threshold, and the line detector is above the threshold.
  - The raptor will turn on the left and right motors at full speed when entering this state.
- **Back:**
  - The raptor is in this state if the light detector is above the threshold, and the line detector is below the threshold, and the backing timer has not expired.
  - The raptor will turn on the left motor at negative half speed and the right motor at negative 75% speed when entering this state (i.e. reverse at an angle).
- **Rotate:**
  - The raptor is in this state if the light detector is above the threshold, and the line detector is below the threshold, and the backing timer has expired.
  - The raptor will turn on the left motor at half speed and the right motor at negative half speed when entering this state (i.e. turn left).

The backing timer makes sure that the raptor backs away from the target for a certain amount of time, and the rotating timer makes sure that the raptor rotates for a certain amount of time before attacking a new part of the fence.

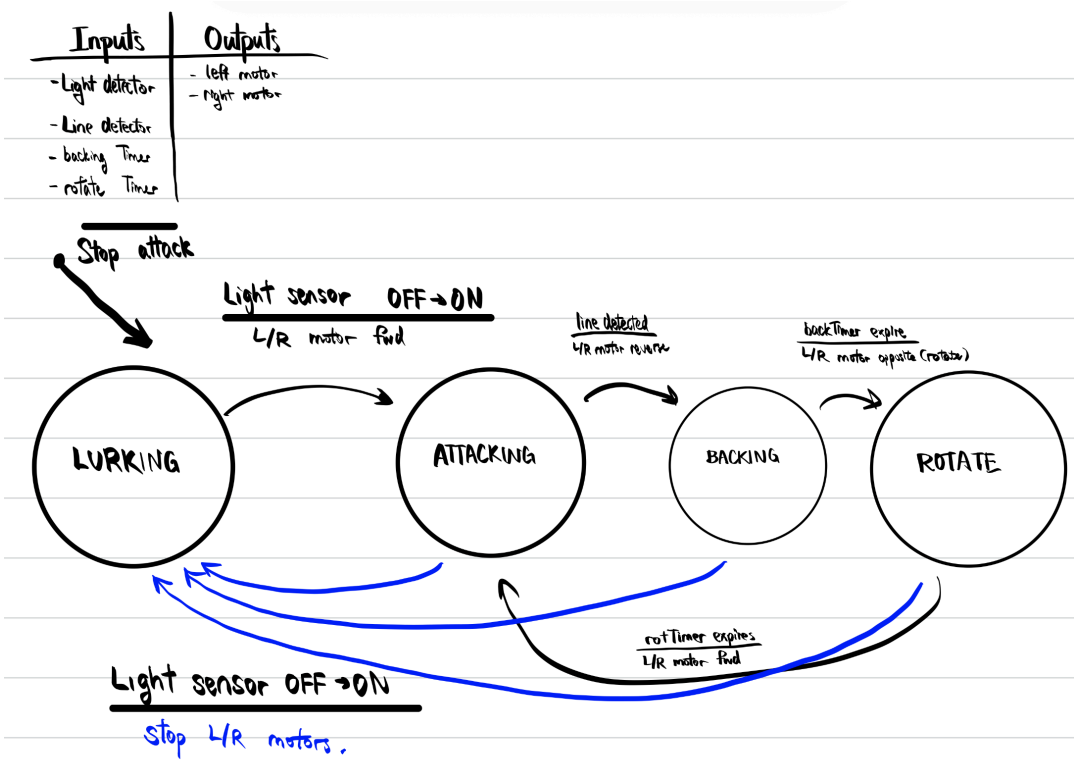

Figure 1: Finite State Machine Diagram of the Raptor

Pseudocode/PDL:

```
// Setup:
setup {
  Start in lurk mode;
}

// Loop:
loop {
  if light below threshold {
    state = lurk;
    motor turn off;
  }
  switch(state) {
    case 1: lurk mode {
      If light detected {
        state = attack;
        put motor in fwd;
      }
    }
    case 2: attack mode {
      If line detected {
        state = back;
        put motor in reverse (at an angle so it doesnt attack same spot);
        set a backing timer;
      }
    }
    case 3: back mode {
      if backing timer expires {
        state = rotate;
        make SPARKI turn (left motor fwd, right motor reverse);
        set a rotating timer;
      }
    }
    case 4: rotate mode {
      if rotating timer expires {
        state = attack;
        put motor in fwd;
      }
    }
  }
}
```

Final Code:

```c
#include <Raptor.h>
#include <SPI.h>
#include <Metro.h>

/*--------------Module Defines----------------------------*/
#define LIGHT_THRESHOLD         75   // *Choose your own thresholds*
                                    // (this will be smaller at night)
#define LINE_THRESHOLD          350   // *Choose your own thresholds*

#define LED_TIME_INTERVAL       2000
#define MOTOR_TIME_INTERVAL     2000
#define BACKING_TIME            3000
#define ROTATE_TIME             2000

#define HALF_SPEED              50
#define FULL_SPEED              100

/*--------------Module Function Prototypes----------------*/
void checkGlobalEventsRaptor(void); // for Q3
void handleMoveForward(void);
void handleMoveBackward(void);
unsigned char TestForKey(void);
void RespToKey(void);
unsigned char TestForLightOn(void);


/*--------------Raptor Function Prototypes----------------*/
void RespToLightOn(void);
unsigned char TestForLightOff(void);
void RespToLightOff(void);
unsigned char TestForFence(void);
void RespToFence(void);
unsigned char TestTimer0Expired(void);
void RespTimer0Expired(void);
void RespBackingTimerExpired(void);
void RespRotateTimerExpired(void);

/*-------------- Simple Test Functions -------------------*/
void motor_on(void);
void motor_off(void);
char get_char(void);
```

```
42  void prt_value(void);
43  uint16_t read_light_lvl(void);
44  void Beep(void);
45  void turn_on_RGB(void);
46  bool test_IR(void);
47  void prt_light_thres(void);
48  void prt_line_thres(void);
49
50  /*---------------State Definitions--------------------------*/
51  typedef enum {
52    STATE_MOVE_FORWARD, STATE_MOVE_BACKWARD,
53    LURK, ATTACK, BACK, ROTATE
54  } States_t;
55
56  /*---------------Module Variables---------------------------*/
57  States_t state;
58  static Metro metTimer0 = Metro(LED_TIME_INTERVAL);
59  static Metro backingTimer = Metro(BACKING_TIME);
60  static Metro rotateTimer = Metro(ROTATE_TIME);
61  uint8_t isLEDOn;
62  uint8_t motorIsFwd;
63  bool backing = false;
64  bool rotating = false;
65
66  /*--------------- Q3.4 Raptor Main Functions ----------------*/
67
68  void setup() {
69    Serial.begin(9600);
70    while(!Serial);
71    Serial.println("Raptors initialized!");
72
73    state = LURK;
74    isLEDOn = false;
75  }
76
77  void loop() {
78    checkGlobalEventsRaptor();
79    Serial.println(read_light_lvl());
80    switch (state) {
81      case LURK:
82        if (TestForLightOn()) RespToLightOn();
83        break;
```

```
84
85      case ATTACK:
86         // default attack mode
87        if (TestForFence()) RespToFence();
88        break;
89
90      case BACK:
91        if (backingTimer.check() && backing) RespBackingTimerExpired();
92        break;
93
94      case ROTATE:
95        if (rotateTimer.check() && rotating) RespRotateTimerExpired();
96        break;
97
98      default:    // Should never get into an unhandled state
99        Serial.println("What is this I do not even...");
100   }
101
102   Serial.println(raptor.LineCenter());
103   Serial.println(raptor.LightLevel());
104 }
105
106 void checkGlobalEventsRaptor(void) {
107   if (TestLedTimerExpired()) RespLedTimerExpired(); // Keep for lab
108   if (TestForLightOff()) RespToLightOff();
109 }
110
111 /*---------------- Module Functions -------------------------*/
112
113 void checkGlobalEvents(void) {
114   if (TestLedTimerExpired()) RespLedTimerExpired();
115   if (TestMotorTimerExpired()) RespMotorTimerExpired();
116 }
117
118 void handleMoveForward(void) {
119   raptor.LeftMtrSpeed(HALF_SPEED);
120   raptor.RightMtrSpeed(HALF_SPEED);
121 }
122
123 void handleMoveBackward(void) {
124   raptor.LeftMtrSpeed(-1 * HALF_SPEED);
125   raptor.RightMtrSpeed(-1 * HALF_SPEED);
```

```
126 }
127
128 uint8_t TestLedTimerExpired(void) {
129   return (uint8_t) metTimer0.check();
130 }
131
132 void RespLedTimerExpired(void) {
133   metTimer0.reset();
134   if (isLEDOn) {
135     isLEDOn = false;
136     raptor.RGB(RGB_OFF);
137   } else {
138     isLEDOn = true;
139     raptor.RGB(RGB_WHITE);
140   }
141 }
142
143 uint8_t TestMotorTimerExpired(void) {
144   return (uint8_t) metTimer1.check();
145 }
146
147 void RespMotorTimerExpired(void) {
148   metTimer1.reset();
149   if (state == STATE_MOVE_FORWARD) {
150     handleMoveBackward();
151     state = STATE_MOVE_BACKWARD;
152   }
153   else {
154     handleMoveForward();
155     state = STATE_MOVE_FORWARD;
156   }
157 }
158
159 uint8_t TestForKey(void) {
160   uint8_t KeyEventOccurred;
161   KeyEventOccurred = Serial.available();
162   return KeyEventOccurred;
163 }
164
165 void RespToKey(void) {
166   uint8_t theKey;
167   theKey = Serial.read();
```

```
  Serial.print(theKey);
  Serial.print(", ASCII=");
  Serial.println(theKey, HEX);
}

/* ---------------- For Q2.6 (estimating thresholds) ----------------------*/

void check_light_thres(void) {
  uint16_t currLightLvl = raptor.LightLevel();
  Serial.print("Current Light Level: ");
  Serial.print(currLightLvl);
  Serial.println(".");
}

void check_line_thres(void) {
  Serial.println("Current Line levels: ");

  Serial.print("Line Left: ");
  Serial.print(raptor.LineLeft());
  Serial.println(".");

  Serial.print("Edge Left: ");
  Serial.print(raptor.EdgeLeft());
  Serial.println(".");

  Serial.print("Line Center: ");
  Serial.print(raptor.LineCenter());
  Serial.println(".");

  Serial.print("Edge Right: ");
  Serial.print(raptor.EdgeRight());
  Serial.println(".");

  Serial.print("Line Right: ");
  Serial.print(raptor.LineRight());
  Serial.println(".");
}

/* ---------------- For Q2.7 (implementation of functions) ----------------------
*/

uint8_t TestForLightOn(void) {
```

```
209   if (raptor.LightLevel() >= LIGHT_THRESHOLD) return 1;
210   else return 0;
211 }
212
213 void RespToLightOn(void) {
214   state = ATTACK;
215   raptor.LeftMtrSpeed(FULL_SPEED);
216   raptor.RightMtrSpeed(FULL_SPEED);
217 }
218
219 uint8_t TestForLightOff(void) {
220   if (raptor.LightLevel() < LIGHT_THRESHOLD) return 1;
221   else return 0;
222 }
223
224 void RespToLightOff(void) {
225   state = LURK;
226   raptor.LeftMtrSpeed(0);
227   raptor.RightMtrSpeed(0);
228 }
229
230 uint8_t TestForFence(void) {
231   if (raptor.ReadTriggers(LINE_THRESHOLD)) return 1;
232   return 0;
233 }
234
235 void RespToFence(void) {
236   state = BACK;
237   backingTimer.reset();
238   backing = true;
239   raptor.LeftMtrSpeed(-1*HALF_SPEED); // reverse at an angle (avoid attacking same
      fence part)
240   raptor.RightMtrSpeed(-75);
241 }
242
243 void RespBackingTimerExpired(void) {
244   backing = false;
245   rotating = true;
246   rotateTimer.reset();
247   raptor.LeftMtrSpeed(HALF_SPEED);
248   raptor.RightMtrSpeed(-HALF_SPEED);
249   state = ROTATE;
```

```
250 }
251
252 void RespRotateTimerExpired(void) {
253   rotating = false;
254   raptor.LeftMtrSpeed(FULL_SPEED);
255   raptor.RightMtrSpeed(FULL_SPEED);
256   state = ATTACK;
257 }
258
259 /* ---------------- For Q2.8 (Simple test functions) ----------------------*/
260
261 // turn motors on
262 void motor_on(void){
263   raptor.LeftMtrSpeed(100);
264   raptor.RightMtrSpeed(100);
265 }
266
267 // turn motors off
268 void motor_off(void) {
269   raptor.LeftMtrSpeed(0);
270   raptor.RightMtrSpeed(0);
271 }
272
273 // get char from keyboard
274 char get_char(void) {
275   return Serial.read();
276 }
277
278 // prt value to serial monitor
279 void prt_value(void) {
280   Serial.println("Hello World");
281 }
282
283 // read light sensor
284 uint16_t read_light_lvl(void) {
285   return raptor.LightLevel();
286 }
287
288 // beep buzzer
289 void Beep(void) {
290   raptor.Beep(100, 5000);
291 }
```

```
292
293 // turn on RGB LED
294 void turn_on_RGB(void) {
295   raptor.RGB(RGB_WHITE);
296 }
297
298 // Test IR sensors
299 bool test_IR(void) {
300   uint8_t triggerState = raptor.ReadTriggers(LINE_THRESHOLD);
301   if (triggerState) return true;
302   else return false;
303 }
```