ME210 Introduction to Mechatronics
Lab 0

Winter 2026
Julia Jiang

Section 2

2.5 LED Blinking Frequency

The LED doesn't blink at 0.5Hz because the time interval of the delay lines in the moving forward and backward functions. While the delay allows the motors to run for some time, it also stops the light from blinking on time. To fix this, I added a timer that expires whenever it's time for the robot to switch from moving forward and backward so the light can blink at 0.5Hz without getting affected by the motor movement.

2.6 Identifying Light and Line Thresholds

LIGHT_THRESHOLD = 0
LINE_THRESHOLD = 0

```
void PrintLightThreshold(LIGHT_THRESHOLD){
  //identify light threshold
  Serial.println(LIGHT_THRESHOLD)
}
void PrintLineThreshold(LINE_THRESHOLD)){
  //identify line threshold
  Serial.println(LINE_THRESHOLD)
}
```

2.7 Given Functions

```
uint8_t TestForLightOn(void) {
  //returns 1 if light is on
  if (raptor.LightLevel() >= LIGHT_THRESHOLD) return 1;
  return 0;
}
uint8_t TestForLightOff(void) {
  //returns 1 if light is off
  if (raptor.LightLevel() < LIGHT_THRESHOLD) return 1;
  return 0;
}
void RespToLightOn(void) {
  state = STATE_ADVANCING;
  return;
}
void RespToLightOff(void) {
  state = STATE_LURKING;
  return;
}
uint8_t TestForFence(void) {
  //returns 1 if there is fence
  if (raptor.ReadTriggers(LINE_THRESHOLD)) return 1;
  return 0;
}
void RespToFence(void) {
  state = STATE_RETREATING;
  return;
}
```

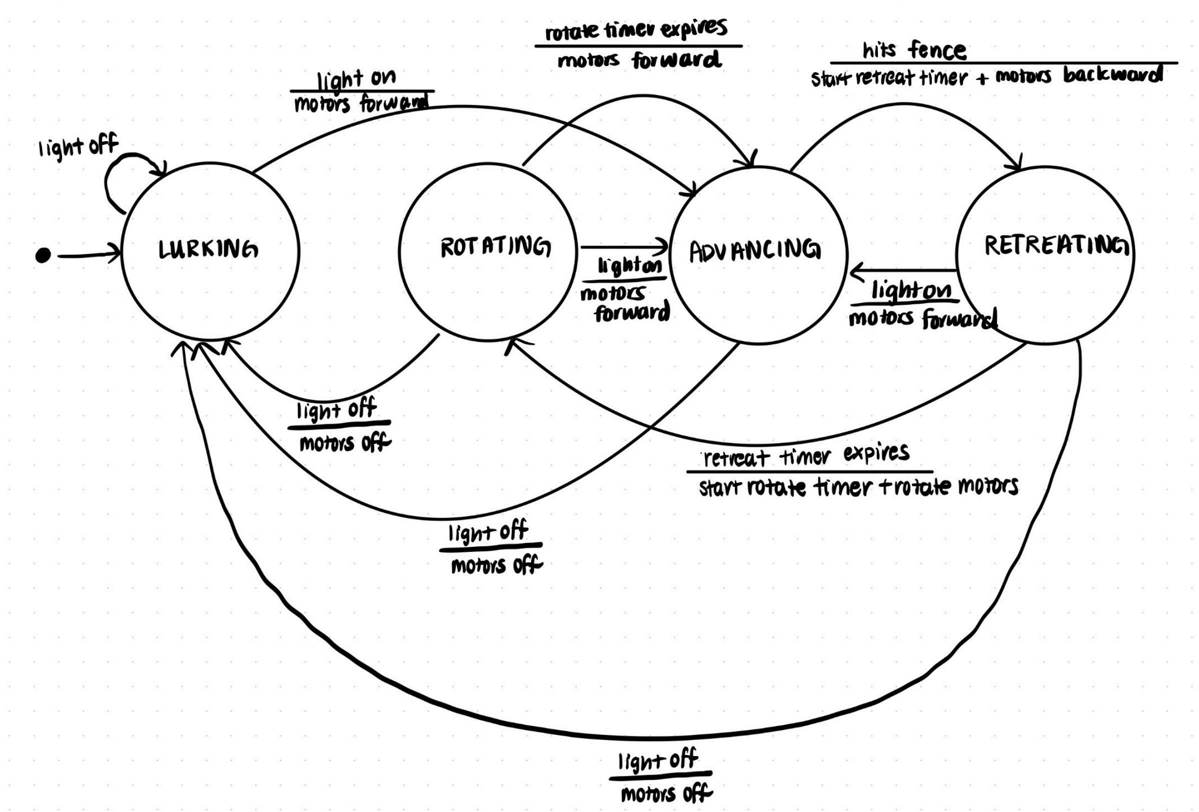
2.8 Simple Testing Programs

```
void TurnMotorOn(){
  Raptor.LeftMtrSpeed(HALF_SPEED);
  Raptor.RightMtrSpeed(HALF_SPEED);
  return;
}
void TurnMotorOff(){
  Raptor.LeftMtrSpeed(0);
  Raptor.RightMtrSpeed(0);
  return;
}
char GetChar(){
  return Serial.read();
}
void PrintValue(){
  Serial.println("Hello World!");
  return;
}
void ReadLightSensor(){
  int16_t light_sensor = Raptor.LightLevel();
  Serial.println(light_sensor);
}
void BeepBuzzer(){
  Raptor.Beep(260, 5000);
  return;
}
void TurnOnLed(){
  Raptor.RGB(200, 100, 150); // R, G, B
  return;
}
// test ir bumpers
void IsLeftLine(){
  trigger_state = Raptor.ReadTriggers();
  return (trigger_state & 0x01);
}
void IsLeftEdge(){
  trigger_state = Raptor.ReadTriggers();
  return (trigger_state & 0x02);
}
void IsCenterLine(){
  trigger_state = Raptor.ReadTriggers();
  return (trigger_state & 0x04);
}
void IsRightEdge(){
  trigger_state = Raptor.ReadTriggers();
  return (trigger_state & 0x08);
}
void IsRightLine(){
  trigger_state = Raptor.ReadTriggers();
  return (trigger_state & 0x10);
}
```

Section 3

3.1

Finite State Diagram



Pseudocode

```
//INITIALIZATION/////////////////////////////////////////
initial state: lurking
//MUST ALWAYS CHECK EVENTS IN LOOP /////////////////
checking global events:
  if light is on: keep advancing/retreating/rotating
  if light is off: motors off
  if rotate timer expires and we are rotating: start advancing
  if retreat timer expires and we are retreating: start rotating
  if we hit a fence: retreat
//ACTIONS/////////////////////////////////////////////
advance: move forward, both motors have same speed
retreat: move backward but not in a straight line(motors at different nonzero
speeds)
rotate: one motor has speed 0 but the other has nonzero speed
```

```
//EVENTS/////////////////////////////////////////
checking if light is on: light on if sensor > light threshold
checking if light is off: light off if sensor < light threshold
checking if we hit a fence: use helper function (read triggers from bottom
sensors and compare to line threshold)
rotate timer expires
retreat timer expires
```

Final Source Code

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

/*---------------Module Defines-----------------------------*/

#define LIGHT_THRESHOLD          50   // *Choose your own thresholds*
                                    // (this will be smaller at night)
#define LINE_THRESHOLD          350   // *Choose your own thresholds*

#define LED_TIME_INTERVAL       2000 //ms
#define MOTOR_TIME_INTERVAL     2000  //time to move forward or backward
#define ROTATE_TIME_INTERVAL    3000 //time to rotate
#define RETREAT_TIME_INTERVAL  3000 //time to retreat

#define HALF_SPEED              50

#define TIMER_0            0
/*---------------Module Function Prototypes-----------------*/
void handleAdvance(void);
void rotate(void);
void handleRetreat(void);
uint8_t TestLedTimerExpired(void);
uint8_t TestRetreatTimerExpired(void);
uint8_t TestRotateTimerExpired(void);
void RespLedTimerExpired(void);
void RespRotateTimerExpired(void);
void RespRetreatTimerExpired(void);
uint8_t TestForKey(void);
void RespToKey(void);
void checkGlobalEvents(void);
uint8_t TestForLightOn(void);
uint8_t TestForLightOff(void);
void ResptToLightOn(void);
void ResptToLightOff(void);
uint8_t TestForFence(void);
void RespToFence(void);

/*---------------State Definitions--------------------------*/
typedef enum {
  STATE_MOVE_FORWARD, STATE_MOVE_BACKWARD, STATE_LURKING,
  STATE_ADVANCING, STATE_RETREATING, STATE_ROTATING
} States_t;

/*---------------Module Variables---------------------------*/
States_t state;
static Metro metTimer0 = Metro(LED_TIME_INTERVAL);
static Metro retreatTimer = Metro(RETREAT_TIME_INTERVAL);
static Metro rotateTimer = Metro(ROTATE_TIME_INTERVAL);
uint8_t isLEDOn;

/*---------------raptor Main Functions----------------*/

void setup() {
  /* Open the serial port for communication using the Serial
     C++ class. On the Leonardo, you must explicitly wait for
   the class to report ready before commanding a println.
  */
  Serial.begin(9600);
  while(!Serial);
  Serial.println("Hello, world!");

  state = STATE_LURKING; //initial state is lurking
  isLEDOn = false;
}

void loop() {
  checkGlobalEvents();
  switch (state) {
    case STATE_ROTATING:
      rotate();
      break;
    case STATE_ADVANCING:
      handleAdvance();
      break;
    case STATE_RETREATING:
      handleRetreat();
      break;
    default: //default state is lurking
      handleLurk();
  }
}

/*----------------Module Functions--------------------------*/
void handleAdvance(void) {
  raptor.LeftMtrSpeed(HALF_SPEED); //move forwards
  raptor.RightMtrSpeed(HALF_SPEED);
}

void rotate(void){
  raptor.LeftMtrSpeed(HALF_SPEED);
  raptor.RightMtrSpeed(0);
}
```

```
void handleRetreat(void) {
  state = STATE_RETREATING;
  raptor.LeftMtrSpeed(-25); //robot doesn't back up in straight line
  raptor.RightMtrSpeed(-1*HALF_SPEED);
}

void handleLurk(void){
  // when we're lurking, we don't want to move
  TurnMotorOff();
  return;
}

uint8_t TestLedTimerExpired(void) {
  return (uint8_t) metTimer0.check();
}

uint8_t TestRetreatTimerExpired(void) {
  return (uint8_t) retreatTimer.check();
}

uint8_t TestRotateTimerExpired(void) {
  return (uint8_t) rotateTimer.check();
}

void RespLedTimerExpired(void) {
  metTimer0.reset();
  if (isLEDOn) {
    isLEDOn = false;
    raptor.RGB(RGB_OFF);
  } else {
    isLEDOn = true;
    raptor.RGB(RGB_WHITE);
  }
}

void RespRotateTimerExpired(void){
  // transition to advancing state
  state = STATE_ADVANCING;
}

void RespRetreatTimerExpired(void){
  // transition to rotating state
  state = STATE_ROTATING;
  rotateTimer.reset();
}

uint8_t TestForKey(void) {
  uint8_t KeyEventOccurred;
  KeyEventOccurred = Serial.available();
  return KeyEventOccurred;
}
```

```
void RespToKey(void) {
  uint8_t theKey;
  theKey = Serial.read();
  Serial.print(theKey);
  Serial.print(", ASCII=");
  Serial.println(theKey, HEX);
}

void checkGlobalEvents(void) {
  //original code for led timer
  if (TestLedTimerExpired()) RespLedTimerExpired();
  if (TestForKey()) RespToKey();

  if(TestForLightOff()) RespToLightOff();
  // when light is on, only start advancing if already in lurking state
  if(state == STATE_LURKING && TestForLightOn()) RespToLightOn();

  // only respond to timer expirations if we are in the state that the timer
is for
  if(state == STATE_ROTATING && TestRotateTimerExpired())
RespRotateTimerExpired();
  if(state == STATE_RETREATING && TestRetreatTimerExpired())
RespRetreatTimerExpired();

  if(TestForFence()) RespToFence();
}

uint8_t TestForLightOn(void) {
  //returns 1 if light is on
  if (raptor.LightLevel() >= LIGHT_THRESHOLD) return 1;
  return 0;
}

uint8_t TestForLightOff(void) {
  //returns 1 if light is off
  if (raptor.LightLevel() < LIGHT_THRESHOLD) return 1;
  return 0;
}

void RespToLightOn(void) {
  state = STATE_ADVANCING;
  return;
}

void RespToLightOff(void) {
  state = STATE_LURKING;
  return;
}

uint8_t TestForFence(void) {
  //returns 1 if there is fence
  if (raptor.ReadTriggers(LINE_THRESHOLD)) return 1;
  return 0;
}

void RespToFence(void) {
  state = STATE_RETREATING;
  retreatTimer.reset();
  return;
}

void TurnMotorOff(void){
  raptor.LeftMtrSpeed(0);
  raptor.RightMtrSpeed(0);
  return;
}
```