```
/*************************************************************
 File:     TheMillennialFalcon.ino
 Contents:
 Notes:    Target: Arduino Leonardo
           Arduino IDE vercv    version: 1.6.7
 History:
 when      who  what/why
 ----      ---  ----------------------------------------
 2016-02-22 FMT  program created
 *************************************************************/

/*---------------Includes----------------------------------*/
#include <Servo.h>

/*---------------Module Defines----------------------------*/
/*----- States -----*/
#define  SEARCHING_FOR_FLAP       0
#define  GOING_TOWARDS_FLAP       1
#define  RELOADING_TOKENS         2
#define  SEARCHING_FOR_BALANCE    3
#define  GOING_TOWARDS_BALANCE    4
#define  DUMPING_TOKENS_SERVO_UP  5
#define  DUMPING_TOKENS_SERVO_DOWN 6
#define  SYSTEM_OFF               7

/*----- Time Constants -----*/
#define  ONE_SECOND           1000
#define  TOTAL_TIME           120000UL
#define  REVERSE_TIME         800 // how long robot reverses when hits obstacle
#define  TURN_TIME            100 // how long robot turns when hits obstacle
#define  STOP_TIME            200 // how long robot stops once locates beacon
#define  FOUND_TIME           1000 // how long robot goes forward once finds
beacon
#define  RELOAD_TIME          1.5*ONE_SECOND
#define  DUMP_TIME            ONE_SECOND

/*----- Speed Constants -----*/
#define  FORWARD_LEFT_SPEED    180
#define  FORWARD_RIGHT_SPEED   180
#define  REVERSE_LEFT_SPEED    80
#define  REVERSE_RIGHT_SPEED   80
#define  TURN_REV_SPEED        90
#define  TURN_FOR_SPEED        160

/*----- Servo Constants -----*/
#define  SERVO_DOWN           120
```

```
#define  SERVO_UP            0

/*----- Other Constants -----*/
#define  FLAP              1
#define  BALANCE             2
#define  NOTHING             0

/*---------------Module Function Prototypes---*/
void SearchingForBeacon(int beacon); // search for FLAP
void GoingTowardsFlap(void); // return for more tokens
void ReloadingTokens(void); // stay still
void SearchingForBalance(void); // search for balance
void GoingTowardsBalance(void); // go to balance
void DumpingTokensServoUp(void); // initialize servo
void DumpingTokensServoDown(void); // retract servo
void SystemOff(void);

void TurnRight(void);
void TurnLeft(void);
void GoForward(void);
void GoBackwards(void);
void Stop(void);
void DeployTokenDispenser(void);

unsigned char TestForFLAPOrBalance(int pin);
unsigned char TestForBumperHit(int bumperPin);
unsigned char TestForBothBumpersHit(int direction);

/*---------------Global Variables-------*/
/*---- Arduino Pins-----*/
int IRPinBack = 4; // IR pin at the left side
int IRPinFront = 2; // IR pin at the right side

int servoPin = 11;
int motorPinDirLeft = 6;
int motorPinEnLeft = 8;
int motorPinDirRight = 3;
int motorPinEnRight = 14;

int bumperPinTopRight = 5;
int bumperPinTopLeft = 12;
int bumperPinBottomRight = 7;
int bumperPinBottomLeft = 13;
int bumperPinTopCenter = 10;

/*---- State Variables-----*/
```

```
int state = SEARCHING_FOR_BALANCE; // begin with 7 tokens

/*----Other Variables-----*/
int frequency = 0; // initialize frequency
Servo servo;

int startTime = 0;

void setup() {
  Serial.begin(9600);

  // Initialize IR pins
  pinMode(IRPinFront, INPUT);
  pinMode(IRPinBack, INPUT);

  // Initialize limit switches
  pinMode(bumperPinTopRight, INPUT);
  pinMode(bumperPinTopLeft, INPUT);
  pinMode(bumperPinBottomRight, INPUT);
  pinMode(bumperPinBottomLeft, INPUT);
  pinMode(bumperPinTopCenter, INPUT);

  // Initialize motors
  servo.attach(servoPin);
  servo.write(SERVO_UP);
  pinMode(motorPinDirLeft, OUTPUT);
  pinMode(motorPinEnLeft, OUTPUT);
  pinMode(motorPinDirRight, OUTPUT);
  pinMode(motorPinEnRight, OUTPUT);

  // Start timer
  startTime = millis();
}

void loop() {
  if (millis() - startTime >= TOTAL_TIME) {
    Serial.println("System off!");
    state = SYSTEM_OFF;
  }
  switch (state) {
    case (SEARCHING_FOR_FLAP): SearchingForBeacon(FLAP); break;
    case (GOING_TOWARDS_FLAP): GoingTowardsFlap(); break;
    case (RELOADING_TOKENS): ReloadingTokens(); break;
    case (SEARCHING_FOR_BALANCE): SearchingForBeacon(BALANCE); break;
    case (GOING_TOWARDS_BALANCE): GoingTowardsBalance(); break;
    case (DUMPING_TOKENS_SERVO_UP): DumpingTokensServoUp(); break;
```

```
   case (DUMPING_TOKENS_SERVO_DOWN): DumpingTokensServoDown(); break;
   case (SYSTEM_OFF): SystemOff(); break;

   default: Serial.println("Something went horribly wrong");
  }
  delay(100);
}

void SearchingForBeacon(int beacon) {
  if (TestForBumperHit(bumperPinTopRight)) {
    GoBackwards();
    delay(REVERSE_TIME);
    TurnRight();
    delay(TURN_TIME);
  }
  if (TestForBumperHit(bumperPinTopLeft)) {
    GoBackwards();
    delay(REVERSE_TIME);
    TurnLeft();
    delay(TURN_TIME);
  }
  if (TestForBumperHit(bumperPinBottomRight)) {
    GoForward();
    delay(REVERSE_TIME);
    TurnLeft();
    delay(TURN_TIME);
  }
  if (TestForBumperHit(bumperPinBottomLeft)) {
    GoForward();
    delay(REVERSE_TIME);
    TurnRight();
    delay(TURN_TIME);
  }
  if (beacon == FLAP) {
    Serial.println("Searching for Flap!");
    if (TestForFLAPOrBalance(IRPinBack) == beacon) {
      Stop();
      state = GOING_TOWARDS_FLAP;
      delay(STOP_TIME);
      GoBackwards();
      delay(FOUND_TIME);
      return;
    }
    TurnRight();
  } else if (beacon == BALANCE) {
    Serial.println("Searching for balance!");
```

```
   if (TestForFLAPOrBalance(IRPinFront) == beacon) {
     Stop();
     state = GOING_TOWARDS_BALANCE;
     delay(STOP_TIME);
     GoForward();
     delay(FOUND_TIME);
     return;
   }
   TurnRight();
  }

}

void GoingTowardsFlap(void) {
  Serial.println("Going towards Flap!");
  if (TestForBumperHit(bumperPinBottomRight) ||
TestForBumperHit(bumperPinBottomLeft)) {
    Stop();
    state = RELOADING_TOKENS;
  } else if (TestForFLAPOrBalance(IRPinBack) == FLAP) {
    GoBackwards();
  } else {
    state = SEARCHING_FOR_FLAP;
  }
}

void ReloadingTokens(void) {
  Serial.println("Reloading Tokens");
  Stop();
  delay(RELOAD_TIME);
  state = SEARCHING_FOR_BALANCE;
}

void GoingTowardsBalance(void) {
  Serial.println("Going towards Balance");
  if ((TestForBumperHit(bumperPinTopCenter) ||
(TestForBumperHit(bumperPinTopCenter) &&
TestForBumperHit(bumperPinTopLeft)) ||
(TestForBumperHit(bumperPinTopCenter) &&
TestForBumperHit(bumperPinTopRight)))) {
    Stop();
    state = DUMPING_TOKENS_SERVO_UP;
  } else if (TestForBumperHit(bumperPinTopRight)) {
    GoBackwards();
    delay(REVERSE_TIME);
    TurnRight();
```

```
    delay(TURN_TIME);
    if (TestForFLAPOrBalance(IRPinFront) == BALANCE) {
      GoForward();
      delay(REVERSE_TIME);
    }
  } else if (TestForBumperHit(bumperPinTopLeft)) {
    GoBackwards();
    delay(REVERSE_TIME);
    TurnLeft();
    delay(TURN_TIME);
    if (TestForFLAPOrBalance(IRPinFront) == BALANCE) {
      GoForward();
      delay(REVERSE_TIME);
    }
  }
  else if (TestForFLAPOrBalance(IRPinFront) == BALANCE) {
    GoForward();
  } else {
    state = SEARCHING_FOR_BALANCE;
  }
}

void DumpingTokensServoUp(void) {
  Serial.println("Dumping tokens servo up");
  servo.write(SERVO_DOWN);
  Stop();
  delay(DUMP_TIME);
  state = DUMPING_TOKENS_SERVO_DOWN;
}

void DumpingTokensServoDown(void) {
  Serial.println("Dumping tokens servo down");
  servo.write(SERVO_UP);
  Stop();
  delay(DUMP_TIME);
  GoBackwards();
  delay(REVERSE_TIME);
  state = SEARCHING_FOR_FLAP;
}

void SystemOff(void) {
  Serial.println("System off!");
  Stop();
}

unsigned char TestForFLAPOrBalance(int pin) {
```

```
  // 50% duty cycle --> T = 2*t_pulse --> f = 500000/t_pulse
  frequency = 500000 / pulseIn(pin, LOW);
  Serial.println(frequency);
  if ((frequency > 4000) && (frequency < 6000)) {
    Serial.println("I see the FLAP");
    return FLAP;
  } else if ((frequency > 800) && (frequency < 1300)) {
    Serial.println("I see a balance!");
    return BALANCE;
  } else {
    return NOTHING;
  }
}

// 0 is front, 1 is back
unsigned char TestForBothBumpersHit(int direction) {
  if (direction == 0) {
    return (digitalRead(bumperPinTopRight) && digitalRead(bumperPinTopLeft));
  } else if (direction == 1) {
    return (digitalRead(bumperPinBottomRight) &&
digitalRead(bumperPinBottomLeft));
  }
}

unsigned char TestForBumperHit(int bumperPin) {
  return digitalRead(bumperPin);
}

void TurnLeft(void) {
  digitalWrite(motorPinEnRight, HIGH);
  digitalWrite(motorPinEnLeft, HIGH);
  analogWrite(motorPinDirLeft, TURN_REV_SPEED);
  analogWrite(motorPinDirRight, TURN_FOR_SPEED);
}

void TurnRight(void) {
  digitalWrite(motorPinEnRight, HIGH);
  digitalWrite(motorPinEnLeft, HIGH);
  analogWrite(motorPinDirLeft, TURN_FOR_SPEED);
  analogWrite(motorPinDirRight, TURN_REV_SPEED);
}

void GoForward(void) {
  digitalWrite(motorPinEnRight, HIGH);
  digitalWrite(motorPinEnLeft, HIGH);
  analogWrite(motorPinDirLeft, FORWARD_LEFT_SPEED);
```

```
  analogWrite(motorPinDirRight, FORWARD_RIGHT_SPEED);
}

void GoBackwards(void) {
  digitalWrite(motorPinEnRight, HIGH);
  digitalWrite(motorPinEnLeft, HIGH);
  analogWrite(motorPinDirLeft, REVERSE_LEFT_SPEED);
  analogWrite(motorPinDirRight, REVERSE_RIGHT_SPEED);
}

void Stop(void) {
  analogWrite(motorPinEnLeft, 0);
  analogWrite(motorPinEnRight, 0);
  digitalWrite(motorPinDirRight, LOW);
  digitalWrite(motorPinDirLeft, LOW);
}
```