MREN 104 - Mechatronics Design Project
Mechatronics and Robotics Engineering Program
Faculty of Engineering and Applied Science, Queen's University, Kingston

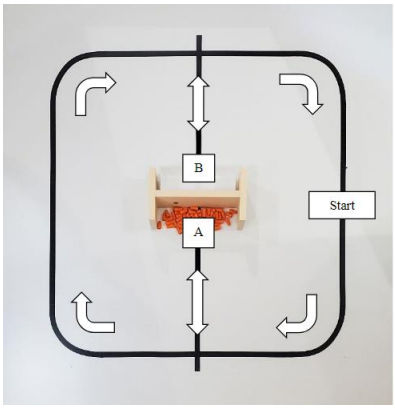
Stage 3 – Development of an Autonomous Loader

Group Number: 4
Student Name: Josh Westlake
Date Report Submitted: December 3, 2023

Summary:
The objective of this project was to develop a fully autonomous mobile robot capable of navigating a predefined path and transporting parts from one location to another without interjection from people. The software component revolved around the development and iteration of the **Group4Stage3** program, which enabled the robot to autonomously follow a marked line using a completely PID control logic system, which focused on manipulating the proportional integral derivative theory in order to track and follow a line whilst picking up and dropping the specified items into the bin. The software, represented by the **Group4Stage3** program, evolved from a specific set of instructions to a dynamic approach based on the robot's status, enhancing adaptability. The iterative development process involved meticulous adjustments to timings, motor inputs, PID constants, and servo motor angles for optimal performance. During the evaluation, the MiniBot demonstrated a 50% success rate in the collection and transportation portion of the assignment. But through further adaptions, the success rate was improved to 80%. Overall, this project established a solid foundation for the intricate world of control systems in the world of practical robotics and highlighted some key considerations that are crucial during the design process when in comes to designing and optimizing an autonomous system.

Hardware:

The hardware setup used in the lab is pictured in **Figure 2** below. This setup comprises various components such as a breadboard, an ItsyBitsy microcontroller, LED indicators, a button, resistors, jumper wires, 3 Reflective Object Sensors in miniature form, 2 DC motors, an infrared distance sensor, and a 350 mAh battery that can be recharged via the USB port on the ItsyBitsy. The Itsy-Bitsy M0 Express microcontroller, operating on a 3.3V logic and built around the ATSAMD21G18 Cortex M0+, has a memory of 256KB flash and 32KB RAM. The detailed wiring and pin assignments on the breadboard are further elucidated in Figure 1. A blue LED signals the user when the 3V rail is operational, while a second red LED and a button communicate active signaling and control to the microcontroller. The robot's mobility is powered by T2 FeeTech FM90 DC motors, under the control of the FeeTech FT-SMC02CH servo motor controller which is linked to the microcontroller. The DFRobotShop Rover Line Following Sensor is equipped with 2 QRE1113GR mini reflective sensors that produce 3 analog voltage outputs. For distance detection, a 10–80cm infrared sensor provides data about the proximity of objects to the Minibot. This sensor responds in 39ms and has a consistent current draw of 33mA. Furthermore, the MiniBot is equipped with 2 extra servo motors which are responsible for the translation and rotation of the lift and curl bucket mechanism. The lab evaluation setup used for testing is depicted in **Figure 1**.



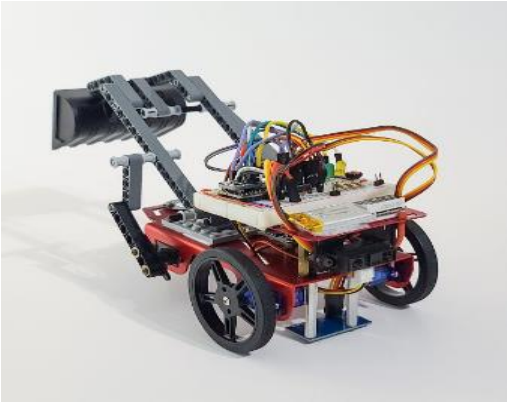


Figure 1: Evaluation Setup **Figure 2:** Hardware Setup

Software:

The program **Group4Stage3** has been included in **Appendix A** and is designed to control a line-following robot using various sensors and control logic. To do this the program initializes and configures the pins for components such as LEDs, a button, a sharp range sensor (SHARP), and left and right line sensors (LSENSOR, RSENSOR). It also assigns the servo motors (leftWheel and rightWheel) to their respective pins and sets constants and variables for control parameters. In the main loop, the program waits for a button to be pressed to initiate its operation. The open loop program a proportional integral derivative control system (PID) that relies on calculating the error between left and right sensor values and implementing the relationship between the integral and derivative of the error values to determine an appropriate offset that needs to be applied to correct for both minor and major adjustments to perfect the miniBot's trajectory. Then the calculated control signal is used to drive the servo motors, enabling the robot to follow a marked line autonomously with much smoother translation than alternative ON/OFF control systems. Overall, this program enables the robot to navigate a predefined path by responding to sensor data using specifically designed control algorithms, which allow the robot to make real-time adjustments to maintain its course. Additionally, the program features 2 distinct run modes. This first indicates to the microprocessor that the miniBot is preparing to pick up the specified objects, while the second state indicates that the robot has objects and is ready to drop them off. In the program, this is done by informing the microcontroller of its current state and updating the status of the robot based on the completion of the previous tasks. In order to facilitate the movement of the bucket servo motor their positions are modulated using a controlled pulse width modulation (PWM) signal, transitioning it incrementally between predefined angular coordinates. This ensures a steady, regulated elevation of the main arm segment. Following this elevation, after a short latency introduced programmatically to ensure temporal separation of actions, the second servo motor swings into action. It controls the bucket segment's angle, first extending it to a pre-designated maximum angular position and then, post a programmatic delay, retracting it, simulating a bucket's operational movement after which servo motor A is commanded via its PWM interface to realign the arm to its origin point, marking the culmination of a full lift-and-lower cycle.

Control Systyem:

Over the course of the project the MiniBot's control system underwent a series of iterations to refine its line-following capabilities. Beginning with the initial "on/off" control system, though straightforward, resulting in extreme movements and suboptimal lap times. Then a transition to a proportional-based control system offered more nuanced control along straights but proved unreliable around corners. Subsequent attempts with a proportional-derivative (PD) control system reduced oscillations but fell short of desired results. The introduction of a proportional derivative (PD) control system enhanced stability and overall performance, yet challenges persisted around sharp corners. Following the painstaking iteration of proportional constants, a concept of combining both aspects of both systems proved to be a very effective strategy. Combining elements of the initial on/off system with the PD control provided a smoother response to adjustments along straight lines while maintaining the reliability of the on/off mechanism, significantly improving lap times and overall performance. The quickest lap times recorded using the on/off PD hybrid system are shown in **Table 1**. Note: the Delta represents the proportional value converted to PWM for robot movement, while the kp and kd values reflect the corresponding factor of which each component of the PD are considered when adjusting the wheel offsets.

Table 1. ON/OFF PD Hybrid

Battery (V)	Delta	forOffset	kp	kd	Time (s)
3.76	13	0.8	0.0047	0.004	43.62
3.67	13	0.8	0.0044	0.004	44.68
3.65	13	0.8	0.0035	0.004	38.64
3.65	13	0.8	0.0035	0.0036	34.45
3.63	13	0.8	0.0035	0.0036	33.23
3.62	13	0.8	0.0032	0.0036	31.32
3.76	13	0.8	0.0034	0.0032	28.34

While these lap times are sufficient the team recognized that the ON/OFF control often clashed with the PD system causing the robot to over-adjust and in some cases completely stop moving when making sharp turns. As a result, it was clear that if there was a way to remove the ON/OFF control system and use solely the proportional system the robot would have much smoother and more effective line-following capabilities. As a result, during stage 3 of the project, many hours were spent exploring the intricacies of Proportional – Integral – Derivative (PID) control systems. This system utilizes the same general concept as the PD system but implements the integral portion which proved to be a very helpful component around the sharpened corners. The addition of the integral system meant that the longer the sensors were sensing the black line the larger the magnitude of a wheel delta would be sent to the wheels allowing the robot to accelerate as it goes around the sharp corners while maintaining the excellent control the PD controls systems implemented. The fastest lap times collected for the PID system can be found in **table 2.** Note: the Delta represents the proportional value converted to PWM for robot movement, while the kp, kd, and ki values reflect the corresponding factor of which each component of the PID are considered when adjusting the wheel offsets.

Table 2. PID Control System

Battery (V)	Delta	forOffset	kp	kd	ki	Time (s)

3.87	10	0.8	0.0047	0.004	0.001	33.91
3.76	11	0.8	0.0047	0.004	0.001	33.76
3.79	11	0.8	0.0035	0.004	0.001	28.65
3.78	12	0.8	0.0035	0.004	0.001	24.45
3.78	12	0.8	0.0035	0.0036	0.001	24.42
3.78	12	0.8	0.0032	0.0036	0.001	24.66
3.78	12.5	0.8	0.0035	0.0036	0.001	22.93
3.77	12.5	0.8	0.0035	0.0032	0.001	23.1
3.77	13	0.8	0.0035	0.0032	0.001	21.58
3.76	13	0.8	0.0034	0.0032	0.001	21.4

As a result, it is clear that the PID system offers a much quicker and overall, more versatile solution to the line-following component of the project.

Further development of the bucket mechanism was explored past stage 2 of the project, which provided the robot with a specific set of motor inputs, delays, and direct instructions providing a systematic but relatively inflexible and overall inconsistent line-following process for the robot. This meant that while the current iteration of the program worked relatively effective for the previous use case, repurposing this program for stage 3 required considerable reconstruction of the program itself. Recognizing the need for adaptability and optimization, to ease the integration in the final stage the program underwent a significant transformation. By introducing two distinct conditions based on the "status of the robot," the program became more dynamic and versatile. The first stage provided the miniBot with a specific set of instructions to follow when the bucket was empty, while stage 2 was triggered after the robot had picked up the items and was ready to drop them off at the drop box. This not only streamlined the line-following process as it minimized the time that the line-following program was ineffective but also laid a more consistent and overall more simplistic approach for the highly meticulous process of adjusting various timings, motor input signals, PID constants, and servo motor angles.

Bucket

From the hardware perspective control of the MiniBot's bucket was done using two servos, which are responsible for the translation and rotation of the lift and curl bucket mechanism. The torque generated in the motors is translated into the linear motion causing the bucket to lift and curl accordingly which is manipulated in the lab to pick up small objects.

To execute this action the program logic ensured precise bucket movements by sending controlled signals to the servo motors using PWM, facilitating actions like lifting, extending, retracting, and returning to the original position, all initiated by a button press. Furthermore, during testing, mechanical singularities were identified. When the bucket's lift parts were fully extended, it reached a point where the arm would fold back on itself. This observation was further confirmed through dedicated simulations aimed at optimizing the bucket mechanism's range and efficiency.

To find the optimal setup for the bucket a series of simulations were run to find the optimal distance, angles, and length of included members to find the most efficient way to maneuver the bucket. Initially, this stage started off by inspecting just the bucket component of the lift mechanism. The resulting simulations have been depicted below in **Figure 4.**

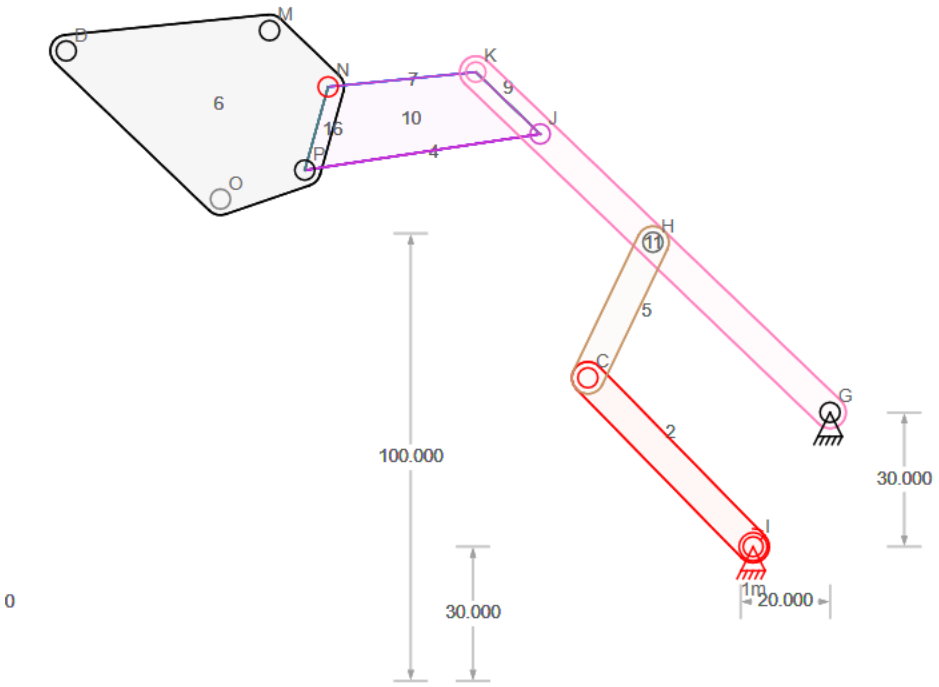


Figure 4. Lift Bucket mechanism Linkage simulation

Similarly, a second simulation focussed on the optimal configuration for the lifting and curling bucket. The results of the simulation have been depicted below. **Figure 5** displays the original configuration, while **Figure 6** showcases the resulting position of each member involved when the mechanism is fully expanded.

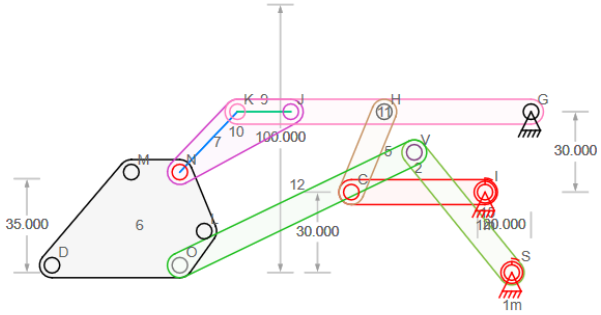


Figure 5. Lift and Curl Bucket mechanism Linkage simulation - Before

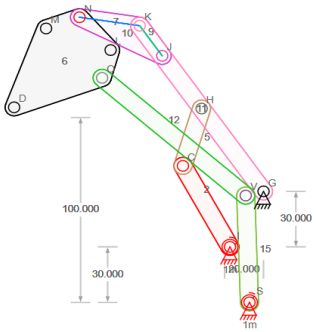


Figure 6. Lift and Curl Bucket mechanism Linkage simulation - After

Results:

Throughout the course of the project, the MiniBot's programming underwent significant transformations, encompassing the evolution of its line-following control system, the

implementation of a bucket mechanism, and the adoption of an open-loop navigation strategy. The culmination of these developments aimed to equip the robot with the versatility needed to accomplish various outlined tasks. Table 1 reveals the outcomes of specific tests conducted during the evaluation procesw. While the MiniBot achieved an 80% success rate, once again falling short of a perfect score, this performance is deemed successful as it consistently completed each task under ideal circumstances. The evaluation stage emphasized the inconsistencies related to voltage variations from the battery. To tackle these issues, a larger cell battery was integrated, extending MiniBot's battery life. Despite these improvements, certain challenges persisted, underscoring the need for further improvements. Further increases in battery size or the introduction of a voltage regulation system are identified as potential avenues to enhance the success rate further.

Table 1. Results for the Autonomous Loader

Test	Battery	Result	Number of Items Collected	Number of Items Delivered
1	3.71V	Items successfully picked up, transported effectively to the drop off, and dropped off in the bin	7	7
2	3.63	Items were successfully picked up, but the robot lost track of the line due to an under-rotation during the pickup process causing the no-drop-off to be completed. Under rotation likely caused by the decreased battery level	4	0
3	4.02	Items were successfully picked up, transported effectively to the drop-off, and dropped off in the bin. Minor over-rotation during the drop off causing it to lose line during exit	8	8*
4	3.82	Items successfully picked up, transported effectively to the drop off, and dropped off in the bin	6	6
5	3.78	Items successfully picked up, transported effectively to the drop off, and dropped off in the bin	5	5

Conclusions:

In conclusion, the team successfully crafted a fully autonomous MiniBot capable of independent navigation and task execution. The evolution of the Group4Stage3 program demonstrated a dynamic shift toward adaptability, with an iterative approach refining its functionality. Achieving an 80% success rate in item collection and transportation underscored the project's accomplishments, while challenges prompted hardware enhancements to address voltage inconsistencies. The integration of a hybrid ON/OFF PD system marked a pivotal improvement, later surpassed by the superior Proportional-Integral-Derivative (PID) control system during stage 3. The intricate hardware setup, depicted in Figures 1 and 2, showcased a sophisticated configuration. This project serves as a valuable exploration into the nuances of control systems,

emphasizing adaptability and optimization for future advancements in the dynamic field of mechatronics and robotics.

Appendix A - Program*:*

```
/************************************************

  MREN104Lab5RangeSensor

  by Josh Westlake, 2023

 This program is intended to combine both the stage 1 and stage 2
  programs together to complete the full evaluation



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

 // Declare servo objects for the motors
Servo leftWheel;
Servo rightWheel;

// Declare servo objects for the motors
Servo myServoA;
Servo myServoB;

// Pins assignments
int GRN = 9;
int BUTTON = 7;     // button pin
int servoPinA = 11; //ServoA
int servoPinB = 12; //servoB

int SHARP = A3;     // sharp range sensor on analog pin 3
int LSENSOR = A2;   // left line sensor on analog pin 1
int RSENSOR = A1;   // right line sensor on analog pin 2

// Integers
int lSensor = 0;     // left sensor value
int rSensor = 0;     // right sensor value
int rangeSensor = 0;
int rangeSensorMv = 0;
int MOTORL = 4;    // left motor signal pin
int MOTORR = 3;    // right motor signal pin
int lasterror = 0;
int button = 0;
int myAngleA1 = 180;    // servo angle 1
int myAngleA2 = 110;    // servo angle 2
int posA;
int bucketAngleA1 = 130; // Servo angle Max
int bucketAngleA2 = 110;   // Servo angle Min
int posB = bucketAngleA2; //position of servo
int state = 1; //status of loading


float stopPulse = 145.5;   // stop pulse for motors (nominal = 150)
float Delta = 11;
float turndelta = 26;
float forOffset = -0.4;    // offset, slows right wheel
float kp = 0.0034;
float kd = 0.0032;
float ki = 0.001;
float reversekp = 0.001;
float integral;

float accelerationFactor = 1;

int THRESHOLD = 1500;

// Program setup
void setup() {


  // Set state of button pin
  pinMode(BUTTON, INPUT);

  // Set state of range sensor pin
  pinMode(SHARP, INPUT);

  // Set line following sensor pins as inputs
  pinMode(LSENSOR, INPUT);
  pinMode(RSENSOR, INPUT);

  //Attatch the motor pins to the servo objects
  leftWheel.attach(MOTORL);
  rightWheel.attach(MOTORR);
  runMotors(0,0);       // stop the motors



  //Setup servo motor
  myServoA.write(110);        // servo A starting position
  myServoA.attach(servoPinA);   // attaches the servo to the servo object
  myServoB.write(posB);
  myServoB.attach(servoPinB);
```

```
  // Start serial communication and set baud rate
  Serial.begin(9600);

} // end of setup
void loop() {
//green LED should blink until button is pressed
do{
  Serial.println(digitalRead(BUTTON));
  flashLED(GRN);
 }while(digitalRead(BUTTON)== LOW);

  do{
    // Read the sensor value
  lSensor = analogRead(LSENSOR);
  rSensor = analogRead(RSENSOR);

  // Read the range sensor value
  rangeSensor = analogRead(SHARP);
  // Map the values into millivolts (assuming 3300 mV
reference voltage)
  rangeSensorMv = map(rangeSensor,0,1023,0,3300);

  // Map the values into millivolts (assuming 3000 mV
reference voltage)
  lSensor = map(lSensor,0,1023,0,3000);
  rSensor = map(rSensor,0,1023,0,3000);

  //initialize variables
  float deltaR;
  float deltaL;
  float error = lSensor -rSensor;
  integral += error;
  float offset = error*kp+integral*ki+(error-lasterror)*kd;
  float reverseoffset = error*reversekp;

//reset the integral component of the PID
if (-100<error<100)
  {
    integral = 0;
  }
//Scooping Condition
if (rangeSensorMv > 1200 && state == 1)
  {
    runMotors(0,0);
    runMotors( Delta-turndelta-forOffset, Delta );
    delay(1250);
    runMotors(0,0);
    delay(10);

    //move servo to ground position
    for( posA = myServoA.read() ; posA <= myAngleA1;
posA ++){
    myServoA.write(posA);
    delay(50);
    }
    //counter to actate line following for 1500 loops
    int counter = 0;
    do{
      linefollow();
      delay(1);
      counter++;
    }while (counter < 1500);

    delay(20);

    //reverse
    runMotors(-Delta-forOffset, -Delta);
    delay(2400);
    runMotors(0,0);
    delay(50);

    //move servo to lifted/angled position
    for (posB = myServoB.read(); posB >= 70; posB --) {
    myServoB.write(posB);
    delay(10);
    }
    //move servo to lifted/angled position
    for (posA = myAngleA1; posA >= myAngleA2; posA --) {
    myServoA.write(posA);
    delay(5);

    }
  //sets a slower acceleration factor and changes the robot
state
  accelerationFactor = 0.4;
  state = 0;

  }
  //drop off condition
  else if(rangeSensorMv > 1250 && state == 0)
  {
    //180 degree turn
    runMotors(0,0);
    runMotors( Delta-turndelta-forOffset, Delta );
    delay(1300);
    runMotors(0,0);
    delay(10);

    //activate line follow for 1500 loops
    int counter = 0;

    do{
      linefollow();
      delay(1);
      counter++;
    }while (counter < 1500);

    delay(20);

    //reverse
    runMotors(-Delta-forOffset, -Delta);
    delay(1300);

    //move the bucket to dumping position
    runMotors(0,0);
    delay(100);
    for (posB = 95; posB <= bucketAngleA1; posB ++) {
    myServoB.write(posB);
    delay(5);
  }

  //return bucket to orignal position
  delay(100);
  for (posB = bucketAngleA1; posB >= bucketAngleA2;
posB --) {
    myServoB.write(posB);
    delay(5);

    }

    //set the acceleration factor and return to scooping state
    accelerationFactor = 0.4;
    state = 1;


  }
  //90 degree turn on double line
```

```
  else if (lSensor > THRESHOLD && rSensor >
THRESHOLD )
  {

    runMotors( Delta, Delta);
    delay(160);
    runMotors(0,0);
    delay(500);
    runMotors( Delta-turndelta, Delta );
    delay(700);
    runMotors(0,0);
    delay(500);
    runMotors(Delta, Delta);
    delay(160);
    accelerationFactor = 0.5;
  }
  //PID program
  if(error > 0){

  deltaR = accelerationFactor*(Delta);
  deltaL = accelerationFactor*(Delta + offset - forOffset);

  runMotors(deltaL, deltaR);
  if (accelerationFactor <= 1)
  {accelerationFactor += 0.0001;}

  }
  else if (error < 0)
  {
  deltaR = accelerationFactor*(Delta - offset);
  deltaL = accelerationFactor*(Delta + offset -forOffset);

  runMotors(deltaL, deltaR);
  if (accelerationFactor <= 1)
  {accelerationFactor += 0.0001;}
  }

  lasterror = error;

  }while(1);


} // end of program loop

// Subroutines



// Flash a single LED and turn others off
void flashLED(int COLOUR)
{
  digitalWrite(COLOUR, HIGH);
  delay(125);
  digitalWrite(GRN, LOW);
  delay(125);
}
void runMotors(float deltaL, float deltaR)
{
  float pulseL = (stopPulse + deltaL)*10;   // calculate length
of pulse
  float pulseR = (stopPulse + deltaR)*10;   // calculate length
of pulse
  leftWheel.writeMicroseconds(pulseR);    // send pulse in
microseconds
  rightWheel.writeMicroseconds(pulseL);   // send pulse in
microseconds
}
//line follow function to activate the PID/line following for a
specified amount of time
void linefollow(){
float stopPulse = 145.5;   // stop pulse for motors (nominal =
150)
float Delta = 11;
float turndelta = 26;
float forOffset = -0.4;    // offset, slows right wheel
float kp = 0.0034;
float kd = 0.0032;
float ki = 0.001;
float reversekp = 0.001;
float integral;

// Read the sensor value
  lSensor = analogRead(LSENSOR);
  rSensor = analogRead(RSENSOR);

  // Read the range sensor value
  rangeSensor = analogRead(SHARP);
  // Map the values into millivolts (assuming 3300 mV
reference voltage)
  rangeSensorMv = map(rangeSensor,0,1023,0,3300);

  // Map the values into millivolts (assuming 3000 mV
reference voltage)
  lSensor = map(lSensor,0,1023,0,3000);
  rSensor = map(rSensor,0,1023,0,3000);


  float deltaR;
  float deltaL;
  float error = lSensor -rSensor;
  integral += error;
  float offset = error*kp+integral*ki+(error-lasterror)*kd;
  accelerationFactor = 0.5;



if(error > 0){

  deltaR = accelerationFactor*(Delta);
  deltaL = accelerationFactor*(Delta + offset - forOffset);

  runMotors(deltaL, deltaR);
  if (accelerationFactor <= 1)
  {accelerationFactor += 0.0001;}

  }
  else if (error < 0)
  {
  deltaR = accelerationFactor*(Delta - offset);
  deltaL = accelerationFactor*(Delta + offset -forOffset);

  runMotors(deltaL, deltaR);
  if (accelerationFactor <= 1)
  {accelerationFactor += 0.0001;}
  }

  lasterror = error;

 runMotors(deltaL, deltaR);
}
```