EE3 Final Report: Team Hulk Dabs

Lab Section 1F
Lucas Wolter: 705-099-851
Bryan Wong: 805-111-517

Introduction and Background

The goal of this project was to program a robotic vehicle to follow a path denoted by a solid black line until it reaches a horizontal bar; then, the vehicle should turn around and retrace the same path before stopping at its starting point, also marked by a horizontal bar. The robotic vehicle used is the TI Robotic Systems Learning Kit, which is controlled by a TI MSP432 LaunchPad microcontroller and contains two motor driven wheels as well as a line following sensor board underneath. The general idea of our design is to obtain input from the sensor board, process this information to determine the location of the solid line and how the vehicle should move to stay on this line, and finally write PWM signals to each motor to either turn the vehicle or continue moving forward.

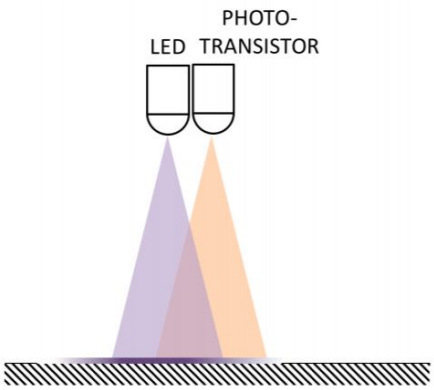

Figure 1: IR emitter/receiver color sensor. The LED emits infrared light which reflects off of nearby surfaces into the phototransistor (receiver) to act as a color sensor. Dark surfaces, such as the line being followed, should give high responses, while light surfaces should give low responses. Credit to Professor Briggs, from “Week 4 Lecture Notes” [1].

The RSLK’s line following sensor board contains eight of these emitter/receiver color sensors. We continually emit light from the IR LEDs by using a digital write command on pins 61 and 45 of the MSP432 Launchpad, which control the even and odd transmitters. The process of reading values from the receivers is more complicated in order to properly distinguish dark surfaces from light surfaces.

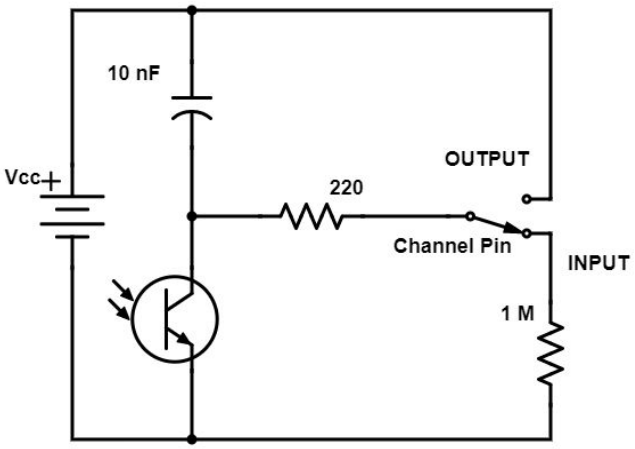

Figure 2: RSLK color sensor schematic. The switch labelled “Channel Pin” allows us to switch between an “Output” and “Input” mode by changing the pin mode for each receiver. In “Output” mode, the circuit acts as a charging RC circuit. Once the 10 nF capacitor is fully charged, the pin is set to “Input” mode and the capacitor discharges through the phototransistor. We then allow the capacitor to discharge and examine the voltage across the receiver. If the phototransistor senses white, it will act as a low-value resistor and the pin will read “LOW”. If the phototransistor senses black, however, it will act as a high-value resistor, increasing the RC time constant and causing the pin to read “HIGH” for longer. This schematic of the phototransistor receiver system was created with Digi-key’s Schemeit [2].

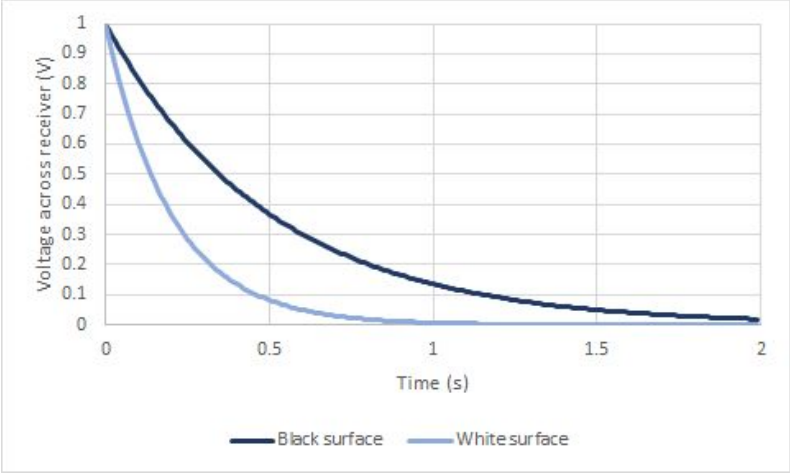

Figure 3: Receiver voltage as capacitor discharges. Surfaces of different hues change the voltage across the receiver at different rates; values shown are arbitrary and are not based on real measurements. In order to accurately sense a black line, a time constant must be found after which the voltage for a white surface is small enough to give a low response while the voltage for a black surface is still large enough to give a high response. Graph based on presentation given by Rahul Iyers titled “Rahul Iyer (Mentor) EE3 S19 Presentation - Intro to Arduino and Energia Programming File” [3].

```
void sampleIR(int readings[8]) {
  // set receivers to output mode
  for(int i = 0; i < 8; i++) {
    pinMode(receivers[i], OUTPUT);
    digitalWrite(receivers[i], HIGH);
  }
  // wait for capacitors to fully charge
  delayMicroseconds(RC_CHARGE);
  // set receivers to input mode
  for(int i = 0; i < 8; i++) {
    pinMode(receivers[i], INPUT);
  }
  // wait until readings become accurate
  delayMicroseconds(RC_DISCHARGE);
  // read values from receivers
  for(int i = 0; i < 8; i++)
    readings[7-i] = digitalRead(receivers[i]);
}
```

Figure 4: IR sampling code excerpt. In our code, receivers are first set to output long enough for the capacitors to fully charge. Then the receivers are switched to input; after waiting long enough for white surfaces to read low and black surfaces to read high, responses are taken with a digital read command.

Testing Methodology

Test Setup

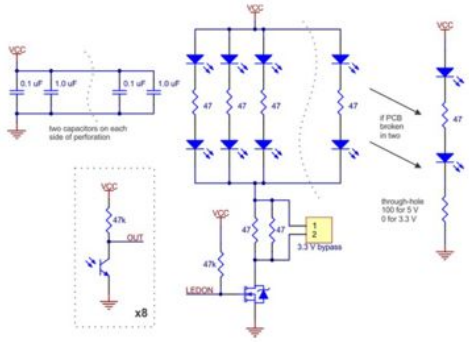

Figure 5: RSLK sensor board schematic. The sensor board consists of eight IR transmitter/receiver pairs; each one contains a capacitor for charging and discharging timer capabilities. One FET transistor is used to turn on all IR LED emitters. Credit to Jason Rubadue, from “UCLA RSLK Intro slides” [4].

Although the TI MSP432 RSLK came mostly assembled, there were a few minor steps we took before undergoing testing. In order to prevent shorting the RSLK when it was plugged into the computer via micro USB, we removed the 5V jumper. The RSLK’s color sensing system operates using an array of 8 IR emitters and receivers. If a white-colored surface is in close enough proximity, emitted IR light is reflected into the receiver. The 8 emitters can be enabled in sets of 4, with odd numbered and even numbered emitters each having their own pin. Each of the 8 receivers has its own pin and corresponding RC circuit; by switching between input and output pin modes with calibrated RC charge/discharge time constants, we can detect whether

the object in front of it is black or white. Placing the sensors on top of a white sheet of paper with black markings to calibrate these time constants was a large portion of our initial testing.

```
sampleIR(readings);
double error_Horiz = // positive value = need to turn right
     -9 * readings[1] -
     4 * readings[2] -  1 * readings[3] +
     1 * readings[4] +  4 * readings[5] +
     9 * readings[6];

if(readings[5] == 0 && readings[6] == 1)
  error_Horiz += 12;
if(readings[2] == 0 && readings[1] == 1)
  error_Horiz -= 12;
```

Figure 6: Error calculation. In order to calculate the error of our system (how far off the line the RSLK is), we took a weighted sum of each IR receiver's reading. A positive error value implies we are veering to the left of the line and a negative error value implies we are veering to the right. If we are perfectly centered on the track, the weighted reading of the left center and right center receivers cancel out and leave us with an error value of 0. We also took precautions for if the car moved off the track so much that only one sensor picked up the track line; this would result in the highest possible error value. In our final product, we disabled the outermost receivers, as they were behaving inconsistently.

```
const int SAMPLESIZE = 10;

class queue {
  public:
    queue() {
      for(int i = 0; i < SAMPLESIZE; i++) {
        array[i] = 0;
      }
    }
    double pushpop(double newVal) {
      double temp = array[pointer];
      array[pointer] = newVal;
      if(++pointer == 10)
        pointer = 0;
      return temp;
    }
  private:
    double array[SAMPLESIZE];
    int pointer = 0;
};
```

Figure 7: Derivative control code excerpt. In order to implement derivative control, we created our own queue class to hold error values from the 10 previous iterations. This was necessary, as the error differential between consecutive iterations is trivial due to the fast sampling rate of the MSP432. In each iteration, the pushpop() function is called, inserting the newest error reading and returning the error reading from 10 iterations ago. The returned error

value is then used to calculate the speed of the car, smoothing out the oscillation from our proportional control.

The RSLK's left/right motors operate using 3 pins each: a "sleep" pin that is set on HIGH to enable it, a directional pin that controls whether movement is forward/backward, and a PWM pin that controls the speed of the motor. Our motor drive subsystem applied a PD control structure to keep the RSLK on the track. Each of the sensor input values were weighted, with outer ones getting higher values than inner ones and left sensors getting negative values; these were summed up to obtain a horizontal error value. This error value was multiplied by a proportional constant, subtracted from the speed of the right wheel, and added to the speed of the left wheel. With the use of a circular queue of size 10, a differential error value was taken by subtracting the horizontal error 10 iterations ago from the current horizontal error; this was multiplied by a derivative constant and also subtracted from the right wheel speed and added to the left wheel speed.

How the tests were conducted

In order to determine the charging and discharging time constants, we added serial print statements to our IR sampling code. We placed the RSLK on a sheet of paper with a black line down the middle, simulating the track, making sure that the sensor board was centered over the line. At first, the charging time constant was kept at 100 microseconds because there was no maximum cap on this value. The discharging time was changed many times while slowly moving the RSLK across the black line and observing what the IR receivers were reading with the print statements. Once an optimal discharging time was found, the charging time was reduced until a minimum working value was found.

The proportional and derivative constant values used in the motor drive subsystem were found by continually changing their values and running the RSLK on a practice track with a straight section and a curved section. First, the derivative constant was set to zero so only the proportional control was acting on the vehicle; an optimal value was found through continual changes to only the proportional constant. At this point, the vehicle would oscillate greatly but still complete the practice track successfully. Next, this value was held the same while the derivative constant was continually changed until an optimal value was found. This would dampen the oscillations of the vehicle to travel more smoothly. Finally, small changes were made to both values until the vehicle moved with optimal precision.

Data Analysis

Time constant	Minimum working value (μs)	Optimal working value (μs)	Maximum working value (μs)
Charging time	3	5	n/a
Discharging time	765	775	790

Figure 8: Working time constant values. After testing many different values, the minima and maxima shown were determined. However, these values were somewhat inconsistent; some of the sensors irregularly gave false values. Therefore, optimal values, which were found to be the most consistent, were used in the final version of code.

Proportional constant, K_P	Derivative constant, K_D	Behavior
1.0	0.0	Unable to follow curved section of the track
2.0	0.0	Inconsistently follows curved section on outer edge
4.0	0.0	Overshoots turn on curved section, unable to continue
3.0	0.0	Successful on curve, extreme oscillations on straight section
3.0	0.5	Minimally less oscillations on straight section, hardly noticeable
3.0	1.5	Oscillations removed but unable to complete curved section
3.0	1.0	Very few oscillations, inconsistently completes curved section
3.2	1.0	More oscillations but consistently completes curved section
3.2	1.2	Fewer oscillations, still completes curved section

Figure 9: Effects of changing PD constants. After testing many different combinations of proportional and derivative constant values, the optimal values were found to be 3.2 for proportional and 1.2 for derivative. This gave the best combination of minimal oscillations and ability to follow on curved track.

Test Data Interpretation

A minimum and maximum value for the discharging time were found to be 765 μs and 790 μs, however the receivers' accuracy was somewhat inconsistent at these extreme values. At the minimum, outer receivers 0 and 7 sometimes gave high responses when positioned over a white surface; receivers 1 and 6 did so as well, but at a much less frequent rate. At the maximum, receivers 3 and 4 sometimes gave low responses when positioned over the black line. Both of these irregular results are not unusual because they match the expected behavior when the discharging time is too short or too long. However, the fact that certain receivers were responding differently than others was somewhat surprising, but can be attributed to the speculation that each receiver is calibrated slightly differently. To ensure consistency in values read, an optimal discharging time of 775 μs was found between the minimum and maximum.

A minimum value for the charging time was found to be 3 μs; a maximum value does not exist because overshooting the charging time only means the circuit rests idly after the capacitor is fully charged. This minimum value was however somewhat inconsistent, causing receivers to

give low responses when positioned over the black line on a few occasions. This makes sense because a short charging time would leave the capacitor below full charge, meaning it would discharge more quickly and give low responses more often. To fix this issue, an optimal charging time of 5 μs was found and used instead.

Testing out many different combinations of proportional and derivative constant values on the PD motor control gave a variety of different outcomes. When derivative control was nonexistent, small proportional values caused an inability to follow the curved section of track; large proportional values caused the RSLK to overshoot turns without reliable recovery. The optimal form of proportional control occurred at a value of 3.0, for which the vehicle successfully completed the curved section consistently but experienced extreme oscillations on the straight section. Adding derivative control successfully dampened the oscillations but sometimes prevented the vehicle from completing the curved section. Small values had little to no effect on the RSLK's movement, while large values completely removed oscillations at the expense of the vehicle's ability to complete the curved section. At a value of 1.0, the vehicle moved with few oscillations but was still somewhat inconsistent on the curve. Adjusting the proportional constant to 3.2 and the derivative constant to 1.2 improved the curve consistency but reintroduced slightly more oscillations; however, this was found to be the most optimal combination.

Results and Discussion

Test Discussion

On Race Day, our vehicle performed as expected from testing. The algorithm and constants used to sample values from the IR sensors were successful in providing accurate values to the motor drive control subsystem. The proportional and derivative constant values used allowed the RSLK to successfully follow the curved sections of track and oscillate minimally on the straight sections. The car performed optimally on its first run, and no further adjustments were necessary.

Race Day Discussion

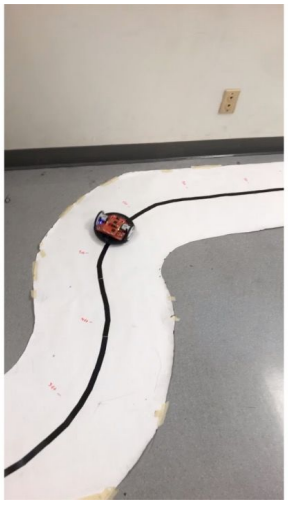

Figure 10: Race Day Run. (video link here) On race day, Hulk Dabs performed just as we had prepared, finishing in about 35 seconds and taking the class lead for 2 minutes.

On race day, our car, aptly named *Hulk Dabs*, was the fourth to hit the track. We did not need to make any adjustments, as we successfully completed the track on the first try. Although its PD control system was not the smoothest, *Hulk Dabs* made up for it with raw power and speed, finishing with a time of about 35 seconds and even taking the title of fastest time for a few minutes. In terms of limitations, *Hulk Dabs* successfully and consistently carried out the given task of navigating the track, turning around, navigating the track backwards, and stopping. However, its PD system was not well optimized, causing it to heavily oscillate as it alternated between accelerating/decelerating each motor.

Given more time, we would have tried to pinpoint the best possible PD values and supplement this with a track recovery system if it ever came completely off the track. We also had trouble with inconsistent outer IR receiver readings that caused us to disable the outermost receivers; if given more time, we would try to troubleshoot these to get more accurate readings. Hopefully, with these further optimizations, we would be able to maximize *Hulk Dabs'* speed and achieve a better race time.

Conclusions and Future Work

Our autonomous line-following car, *Hulk Dabs*, was successful in completing the task we set out to do. It was very consistent in completing the track without straying off the path as well as performing the turning and stopping actions. However, its speed was only decent and our PD system had far too much oscillation, shortcomings that can be further improved in future iterations.

Ultimately, this project was an enriching, hands-on way to learn about complex control systems in the context of an autonomous vehicle. We also got valuable experience working with microcontrollers, exploring the capabilities of the TI MSP432. The project reinforced many concepts, such as RC circuits and their time constants in the IR emitter/receiver system, and PWM in the motor control system. From a programming standpoint, we were able to work with a circular queue and gain experience organizing a large, multi-faceted program. From a teamwork perspective, this project was a great way to work with others over multiple weeks, formulating a plan to tackle a large-scale challenge. By making slight improvements every week, we were able to successfully complete the track.

If we had more time, one direction we could consider taking the project is to recreate the autonomous car using our own hardware, rather than using the RSLK. This would hopefully allow us to get a better understanding of the holistic system, as we can see from a hands-on perspective how each component works. Here are some other various ideas for extension projects we had:

1. Utilize multi-threading to play music as our car drives
2. Use IR sensors to autonomously detect and avoid walls
3. Have the car respond to sound and commands, i.e. stop, drive, turn commands.

We could test each of these options by creating courses with appropriate obstacles (for example: walls, slow zones, etc.) and trying to successfully run them under a specified time limit, much like in our project for EE3.

References

[1] Briggs, Dennis. Week 4 Lecture Notes. UCLA Department of Electrical and Computer Engineering. (2019)

[2] Digi-key. Scheme-it. www.digikey.com/schemeit/project.

[3] Iyer, Rahul. Rahul Iyer (Mentor) EE3 S19 Presentation - Intro to Arduino and Energia Programming File. UCLA Department of Electrical and Computer Engineering. (2019)

[4] Rubadue, Jason. UCLA RSLK Intro slides. Texas Instruments. (2019)