

BUILDING KILOBOTS AND REVISING KILOBOT DESIGN FOR IMPROVING THE
OPTICAL RESPONSE

A thesis presented to the faculty of the Graduate School of
Western Carolina University in partial fulfillment of the
requirements for the degree of Master of Science in Technology

By
Anik Tahabilder

Supervisor: Dr. Yanjun Yan
School of Engineering + Technology

Committee Members:
Dr. Martin Tanaka, School of Engineering + Technology
Dr. Paul Yanik, School of Engineering + Technology
Dr. Peter Tay, School of Engineering + Technology

April 2020

©2020 by Anik Tahabilder

This work is dedicated to my parents
for their endless love and support.

ACKNOWLEDGEMENTS

I would first like to acknowledge my adviser, Dr. Yanjun Yan, for her patience, kindness, and most of all her assistance in completing this thesis. I would like to thank Dr. Martin Tanaka, Dr. Peter Tay, and Dr. Paul Yanik for serving on my thesis committee.

CONTENTS

ABSTRACT.....	vii
CHAPTER1: INTRODUCTION	1
1.1 Objectives	3
1.2 Significance of the study.....	3
CHAPTER 2: Background.....	5
2.1 Swarm Robotics	5
2.2 Kilobot System.....	5
2.2.1 Kilobot	6
2.2.2 Kilobot Charger	9
2.2.3 Bootloader Programmer.....	9
2.2.4 Overhead Controller.....	11
2.3 Programming Environment.....	12
2.4 Shape Formation	13
2.4.1 Subtractive Shape Formation.....	13
2.4.2 Additive Shape Formation	15
CHAPTER3: Methodology.....	16
3.1 Building and Challenges	16
3.2 Light Based Operation	21
3.3 Insufficient Memory Issue	24
3.4 Debugging Model	26
3.5 Kilobot Version Update	27
CHAPTER 4: RESULTS.....	31
4.1 The first revised design (version 1.2)	31
4.2 The second revised design (version 1.3).....	35
4.3 The third revised design (version 1.4)	39
CHAPTER 5: CONCLUSION	46
References.....	48
APPENDIX.....	51

LIST OF TABLES

Table 1. Properties of WCU Kilobot	8
Table 2. Ambient light sensing using the Arduino Uno	22
Table 3. Ambient light sensing using the potentiometer of a Kilobot	23
Table 4. Microcontroller property comparison.....	25
Table 5. Successful move-away-from-light test result using $R_{35}=604\text{ K}\Omega$ out of 5 trials.....	36
Table 6. Successful move-away-from-light test result using $R_{35}=11\text{ K}\Omega$ out of 5 trials.....	36
Table 7. Ambient light sensitivity for small distance increment	41
Table 8. Key properties of ATmega1284	42

LIST OF FIGURES

Figure 1. Kilobot and OHC.....	6
Figure 2. Kilobot components.....	7
Figure 3. WCU Kilobot version 1.1.....	8
Figure 4. Kilobot charger.....	9
Figure 5. Kilobot Bootloader Programmer.....	10
Figure 6. Modified cable connection.....	10
Figure 7. Kilobot Bootloader interface with Kilobot.....	10
Figure 8. Overhead controller (OHC).....	11
Figure 9. Kilogui interfacing between the OHC and the computer.....	12
Figure 10. Kilobotics program editor.....	13
Figure 11. Subtractive shape-formation.....	14
Figure 12. Additive Shape-formation.....	15
Figure 13. The Stencil that we used.....	16
Figure 14. The reflow oven we used.....	17
Figure 15. Kilobot board (SMD mounted).....	17
Figure 16. Kilobot board microscopic view before debugging.....	18
Figure 17. Kilobot board microscopic view after debugging.....	19
Figure 18. Kilobot board view of inaccessible parts.....	20
Figure 19. Ambient light circuit using Arduino Uno.....	21
Figure 20. Sensor reading vs. light intensity using Arduino Uno.....	22
Figure 21. Kilobot with a potentiometer for ambient light test.....	23
Figure 22. Sensor reading vs. light intensity using modified Kilobot.....	24
Figure 23. Workspace in Altium Designer.....	27
Figure 24. Schematic of Kilobot version 1.1.....	31
Figure 25. Schematic of Kilobot version 1.2 (Microcontroller Unit).....	32
Figure 26. Schematic of Kilobot version 1.2 (Power Unit).....	32
Figure 27. PCB layout of Kilobot version 1.2.....	33
Figure 28. 3D view of PCB layout of Kilobot version 1.2.....	34
Figure 29. Ambient Light Circuit update in second revised (WCU Kilobot version 1.3).....	35
Figure 30. Move away from light test using old Kilobot (Starting).....	37
Figure 31. Move away from light test using old Kilobot (Stopped due to saturation).....	37
Figure 32. Move away from light test using new Kilobot (Starting).....	38
Figure 33. Move away from light test using new Kilobot (Turning).....	38
Figure 34. Move away from light test using new Kilobot (Moving Away).....	38
Figure 35. Measuring light intensity using Lux Light Meter.....	39
Figure 36. Small distance light sensitivity test for adding the second sensor.....	40
Figure 37. Schematic of two ambient light sensors.....	41
Figure 38. Schematic of the new MCU ATmega1284.....	42
Figure 39. Schematic of the of Kilobot version 1.4 (Microcontroller Unit).....	43
Figure 40. Schematic of the of Kilobot version 1.4 (Power Unit).....	43
Figure 41. PCB layout of Kilobot version 1.4.....	44
Figure 42. 3D view of Kilobot version 1.4.....	45

ABSTRACT

BUILDING KILOBOTS AND REVISING KILOBOT DESIGN FOR IMPROVING THE OPTICAL RESPONSE

Anik Tahabilder, M.S.T.

Western Carolina University (Apr 2020)

Director: Dr. Yanjun Yan

Inspired by the emergent behavior of swarms, we want to eventually use a distributed self-organizing swarm of robots for shape formation. To verify the idea using real robots in the experiment, we need to first build more Kilobots to enlarge our repository of Kilobots. Kilobot is a kind of small robot with a 33-mm diameter that was originally designed by Harvard in 2012 and redesigned at WCU with the simplified building process in 2016 (WCU Kilobot version 1.1). Based on the earlier design, we have redesigned the Kilobots further (with three revisions in version 1.2, 1.3, and 1.4). This research work describes the challenges and solutions in building and debugging Kilobots, as well as the planned shape formation operation. Kilobots are built in-house using reflow soldering for surface mount components and hand soldering for through-hole components. A systematic debugging procedure, as well as the most commonly seen issues and their solutions, are described based on our building and testing experience. The WCU Kilobot version 1.1 was designed in PADS, and yet we no longer had the license in PADS. Therefore, we redid the schematics and PCB layout in Altium Designer, and enlarged the spacing between the crowded components, in WCU Kilobot version 1.2. Although the design of version 1.2 was nearly the same as in version 1.1 with only added spacing, it was redone in Altium Designer that we could continue to maintain a license, and hence our later revisions were possible. In shape

formation, a phototaxis movement (moving away from light) is the driving force in the large-scale reductive approach, and yet the original Kilobot design allows such movement only in a dark room because of the ambient light sensor output is saturated at a low illumination level. An experiment was conducted to examine the saturation of sensor reading at increasing lux levels with different phototransistor's emitter resistances, and a new resistance value of emitter resistance was proposed and implemented in our Kilobots (version 1.3), to ease the experiment lighting condition, making it more lenient and convenient than before, even at daylight. An earlier capstone experiment in 2018-2019 seemed to indicate that the flash memory of ATmega328P, the microcontroller on the Kilobot, was not enough to handle the calculation when more than three Kilobots with known or calculated locations were used for multilateration-based locationing for the next robot that needed to calculate its location. To address this issue, we have updated the design of Kilobot to replace its ATmega328p (with 32 Kbytes memory) microcontroller with ATmega1284, which has 128 Kbytes of flash memory for programming (version 1.4). In addition, we also inspected the feasibility of installing two ambient light sensors at opposite sides of the Kilobot and found the version 1.3 was more sensitive than version 1.1 to provide distinctive readings even at a distance increment of one Kilobot diameter, which meant that the Kilobot could easily tell the direction of the light with two sensors. Given the new microcontroller in version 1.4 with more IO channels, we further revised it to add a second ambient light sensor, which will help to give us more control on the Kilobot when they perform a light based movement, such as in shape formation.

CHAPTER1: INTRODUCTION

The role of robotics is to take on the jobs that are repetitive, requiring high accuracy and precision, or not executable by a human being. The application of robotics has increased the gross industrial productivity to a large extent. The Single Robotic System (SRS) has been applied in industries and research institutions for many complex functionalities. However, there are some situations when swarm robotics will be more convenient than a single robot to apply a group of small robots to perform a complex operation. This type of swarm behavior can be also observed in nature. For example, flocks of birds, schools of fish, and a colony of ants organize themselves to form a unitized cluster to perform complex action. Inspired by nature, researchers have started to integrate groups of small robots to work together to complete a complex task by applying robots that have very basic functionality. The researchers had developed various platforms and programming tools for swarm robotics. In most cases, the small robot communicates only within a limited range requiring an efficient control strategy. Kilobots were initially designed by Harvard University in 2012 and redesigned for construction simplicity at WCU in 2016. It is one of the widely used platforms that researchers have used on both hardware and software design for swarm robotics. The WCU Kilobots lab has explored the control strategy in the Kilombo simulation environment earlier and sponsored multiple capstone projects on scaling up the localization ability of the physical Kilobots. To continue the efforts, we need to build more Kilobots.

Shape formation is a popular task in swarm robotics. In shape formation, the algorithm can be grouped into two main categories, i.e., additive process and subtractive process. In the additive process, materials become adjoined and fused to form the desired shape, or the robots cluster together and move along the boundary of the cluster until they find a position to stay to be

part of the cluster of the desired shape. In the subtractive process, materials are reduced from the initial bulk amount, or the robots are placed densely together initially, and the robots outside of the desired shape move away automatically often driven by a light source in the center. Both processes have pros and cons when it comes to practical application. In the additive process, the individual robot's movement may depend on the rest of the group, and hence it takes a comparatively long time to be aligned. On the other hand, in the subtractive process, a bulk of robots is needed initially to be chiseled away and hence the needed number of robots is big, but it will take much less time to form a shape than in the additive process. One potential solution is to combine both additive and subtractive processes in shape formation so that the desired shape may extend beyond the original cluster boundary, and the robots not inside the shape will move along the cluster boundary to fill in the void instead of simply moving away. In all processes, we need a sizable amount of robots to perform practical swarm experiments, and we also need to understand how Kilobots navigate according to the light source as a building-block movement in shape formation.

In our original version of WCU Kilobot version 1.1 built in 2016 following the Harvard schematics but in a different layout, the robot can perform phototaxis movement only in a dark room by design, which is not so convenient. So, in this research, we focused on building more Kilobots to conduct experiments with more flexibility in the regular room condition than before. As a result, this thesis focused more on hardware development. We have inspected the pros and cons of the earlier design, provided three revisions (versions 1.2, 1.3, and 1.4) so that it can be built and debugged easily in-house while enabling us to conduct experiments in a regular daylight environment.

1.1 Objectives

Kilobot is one of the commonly used robots for swarm robotics tests and developments. Purchasing Kilobots is expensive and it does not give the students a solid experience to understand the Kilobot hardware. In-house building and debugging can help students understand the operations and discover some aspects that they can work with for further development. The main goal of this research is to build and revise Kilobot in-house to support swarm robotics experiments. The main objectives of this research include the following:

- Create guidelines on building and debugging Kilobot in-house based on our building experience.
- Update the design to allow a more flexible experiment environment than before.
- Verify the performance of Kilobot that we built in relevant experiments.
- Increase the number of Kilobots.

1.2 Significance of the study

This research was conducted to scale up the Kilobot construction in-house to eventually carry out the shape formation operation. In the WCU Kilobots lab, subtractive shape formation operation for a simple shape (square) at a small scale (using 16 robots in a 4 by 4 pattern) without using the light source has been explored, and yet we need to scale up the experiment to allow a more complex shape using the light source in the center of the shape as a driving force for the movement. Meanwhile, we are exploring the additive shape formation and to fuse both processes together. For all the potential experiments, we need to enlarge our repository of Kilobots. While building Kilobots in house, the building and debugging experience have been used to update the design that will improve the process as well as the light-sensing based movement. The outcomes of the research are listed below:

- The debugging guidelines summarized from our efforts are helpful for the future building process.
- The robots now have a wider operating range of lighting conditions than before.
- The robots could have more flash memory for large scale operation.
- We could have more control of the robots with an added sensor.

CHAPTER 2: BACKGROUND

2.1 Swarm Robotics

Swarm robotics is the coordination of a multi-robot system, which consists of a large number of simple features with small robots. The collective behavior of insects or animals can be imitated by swarm robots in swarm intelligence. Swarm robotics encompasses the design of the robot and control strategies. A key advantage of swarm robotics is the ability to adapt any team member as needed that reduces the chance of failure due to the unavailability or malfunctioning of any individual during the operation. The definition of swarm robotics by Shahin [1] gives an insight into this field.

“Swarm robotics is the study of how a large number of relatively simple physically embodied agents can be designed such that a desired collective behavior emerges from the local interactions among agents and between the agents and the environment.”

There is a lot of platforms to perform simulation and practical test of swarm robotics like Khepera robot [2], e-puck robot [3], Jasmine robot [4], I-Swarm robot [5], S-Bot [6], Kobot [7], SwarmBot [8], Kilobot [9] and so forth. Kilobot is one of the popular platforms for swarm robotics that was developed at Harvard university to program and experiment with collective behaviors in large-scale autonomous swarms [10]. The WCU Kilobot team [11] poses a modified design of Kilobot that was built and tested at WCU in 2016.

2.2 Kilobot System

Kilobot system consists of Kilobot itself, an overhead controller (OHC), the Kilobot charger, a bootloader programmer, and the programming environment. After a Kilobot is built and debugged, it is charged by a charging station that can charge 25 robots at a time. The bootloader we used is loaded through a USB AVRISP XPII, AVR Programmer, with a modified

cable connection for interfacing with Kilobot. We have used Kilogui, a graphical user interface for interfacing the OHC to the computer, and OHC has been used to program the Kilobots by sending programs via IR communication, as shown in Figure 1. The Kilobot is programmed using a C based programming language, and we have used an online compiler, Kilobotics website[12], maintained by Harvard University to write and compile the program.

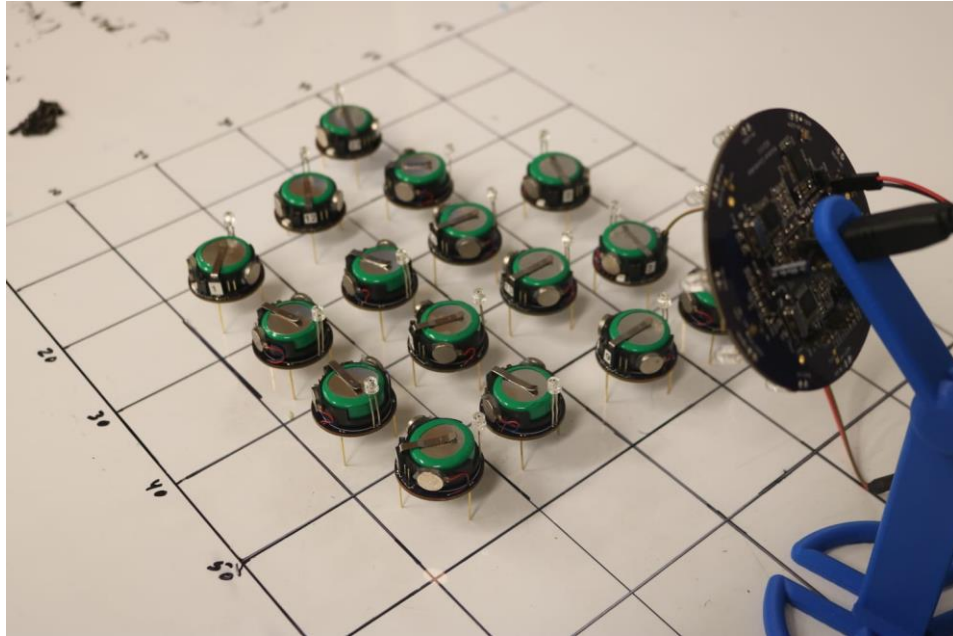


Figure 1. Kilobot and OHC

2.2.1 Kilobot

Kilobots are microrobots with a 33-millimeter diameter that has several basic functionalities. It communicates with other robots by infrared. Its microprocessor can process data and make decisions. It moves based on the slipstick phenomenon using two vibration motors mounted on two out of the three legs. It was designed to be used in large quantities.

Figure 2 shows the parts of a Kilobot [13]:

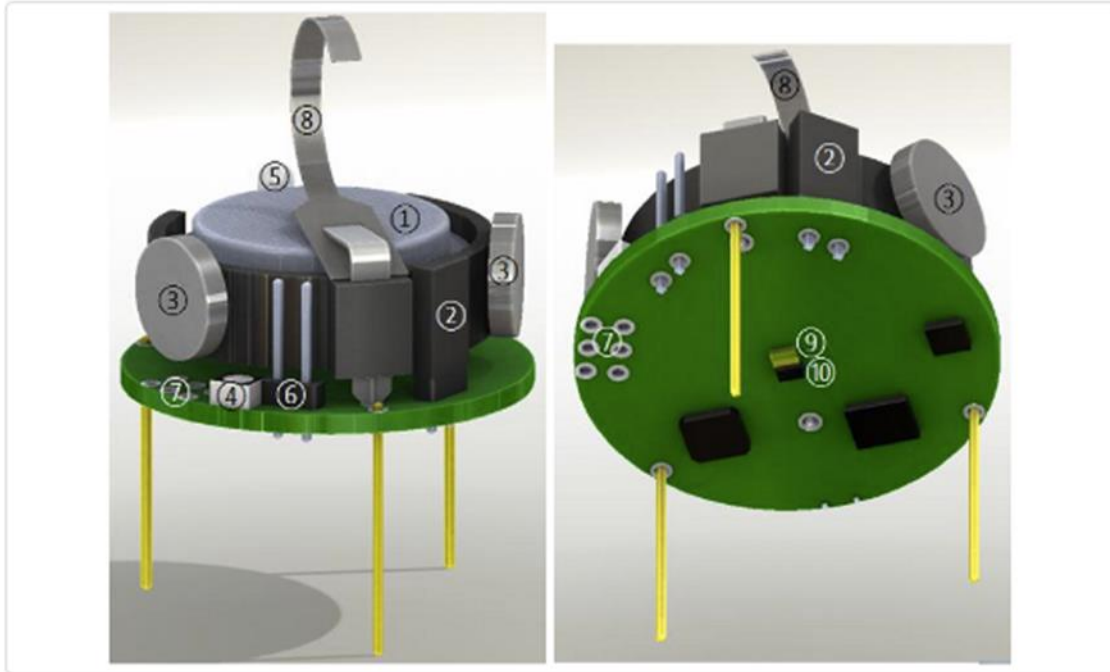


Figure 2. Kilobot components

Part-1 shows a battery to supply power to the robot for its functionality. Part-2 is the power jumper used for turning on and off the robot. Part-3 is the vibration motors to enable the movements. Part-4 shows an RGB LED, which is used to indicate the various status of the robot during its functionality. Part-5 is the ambient light sensor that is used for the light-based operation. Part-6 is the serial output header, which is used to output serial data to the computer for debugging purposes. Part-7 is a direct programming socket that is used to load the firmware to the microprocessor unit of the robot. Part-8 is the charging tab, which has been removed in our WCU revised design as a new charger was designed at that time. Part-9 is the infrared transmitter to transmit IR signals to other robots. The last one, part-10, is the infrared receiver, which is used to receive the infrared signals from other robots or the OHC. Figure 3 shows the previously built WCU Kilobot version 1.1, which used nearly identical schematic to the Harvard design but with a compact layout, and the SMD components were no smaller than 0604 for in-house fabrication. Table1 shows its main specifications.

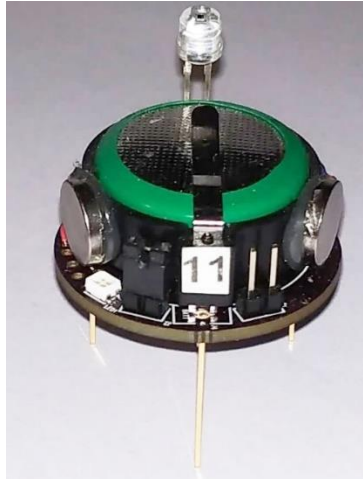


Figure 3. WCU Kilobot version 1.1

Table 1. Properties of WCU Kilobot

Processor	ATmega 328p (8bit @ 8MHz)
Memory	32 KB Flash, 1KB EEPROM
Battery	Rechargeable Li-Ion 3.7V
Charging	Kilobot charger for 25 robots simultaneously
Sensing	1 IR and 1 light intensity
Movement	Forward, Left, Right
Programming	C language with Kilobotics editor
Dimension	diameter: 33 mm, height: 34 mm

In short, a Kilobot contains a microprocessor to make decisions, a rechargeable battery to enable its action, an IR transceiver for sensing and transmitting a signal to other Kilobots or OHC, and a motor vibration based slipstick movement system. We program it using C in an online editor at the Kilobotics website.

2.2.2 Kilobot Charger

To conduct a swarm robotic experiment such as shape formation, we need a lot of robots. Charging all the robots individually is tedious and time-consuming. To simplify the process, a Kilobot charger storage case is used. It is a 3D printed box [14], with 5×5 cylindrical chambers inside for placing each individual robot. The bottom and the top of the box are inserted two copper plates with springs on top to make electrical contact for charging. It can charge 25 Kilobots at a time by using a laptop charger rated at 19.6V, 4.62 Amp. Figure 4 shows the charger docking station for our WCU Kilobots.



Figure 4. Kilobot charger

2.2.3 Bootloader Programmer

We are using the USB AVRISP XPII as the bootloader programmer of our Kilobot. When a Kilobot is built and charged for the first time, it needs to have a bootloader driver. A bootloader programmer is very convenient to load the bootloader file into a Kilobot microcontroller. We are using Atmel Studio to interface with Kilobot using a bootloader programmer. Figure 5 shows the image of the bootloader programmer.

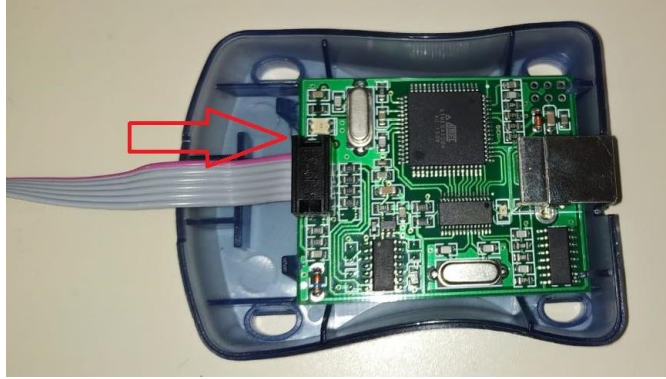


Figure 5. Kilobot Bootloader Programmer

This programmer comes with a serial peripheral interface that is primarily designed for Arduino and some other similar devices. Since the physical orientation of the Kilobot interface port is different, we have modified the interface of cable to establish ISP communication with the Kilobot. Figure 6 shows the modification of cable for connecting the WCU Kilobot. Figure 7 shows the connection of Kilobot with the programmer using our modified cable.

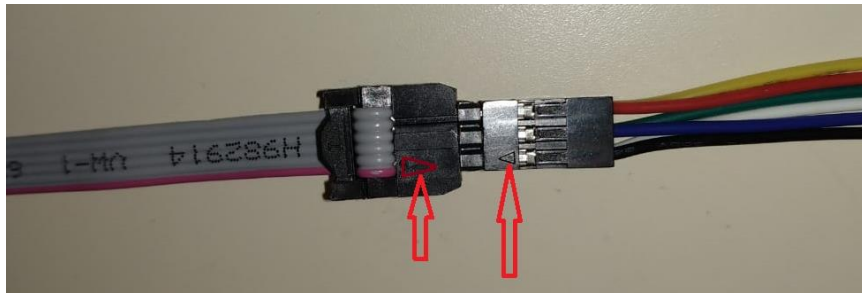


Figure 6. Modified cable connection

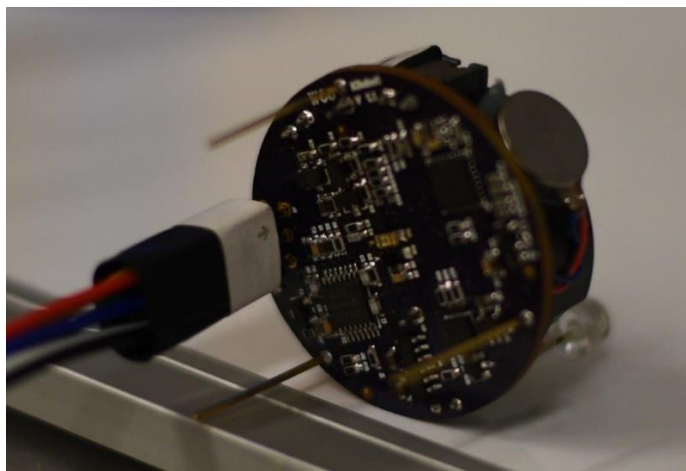


Figure 7. Kilobot Bootloader interface with Kilobot

2.2.4 Overhead Controller

Programming each Kilobot through cable is tedious and nearly impossible for thousands of Kilobots. So, the Harvard University Kilobot team provided the design of an overhead controller for programming Kilobots using infrared signals. WCU built the overhead controller in-house earlier to program all the robots by interfacing them through IR. The coverage area is about a circle of one-meter diameter for programming [15]. Figure 8 shows the overhead controller of the WCU Kilobot system [16]. Note that a serial cable is connected between the OHC and a robot in Figure 8, that is to send serial data back to the computer through the OHC. For programming purpose alone, the IR transmission from the OHC to the robots is wireless.

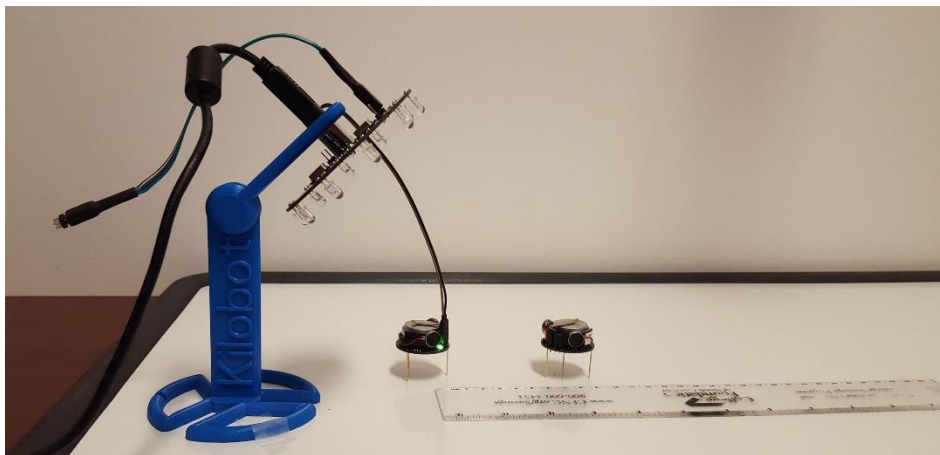


Figure 8. Overhead controller (OHC)

To interface with the overhead controller through a USB cable from the computer, a graphical user interface app called KiloGUI, as shown in Figure 9, is used [16]. KiloGUI can be used to calibrate the Kilobot motors and assign its ID, and to send a program from the computer to the OHC, to distribute it to the robots eventually. This app is user-friendly with a serial reading window. When the robot is connected to the OHC through a serial cable, as shown in Figure 8, KiloGUI can read the data from the robot such as ambient light sensor output or the calculation results during an experiment, which is essential for debugging.

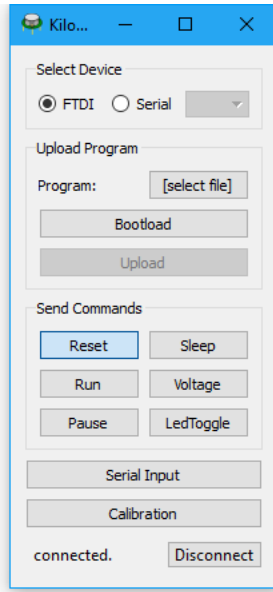


Figure 9. Kilogui interfacing between the OHC and the computer

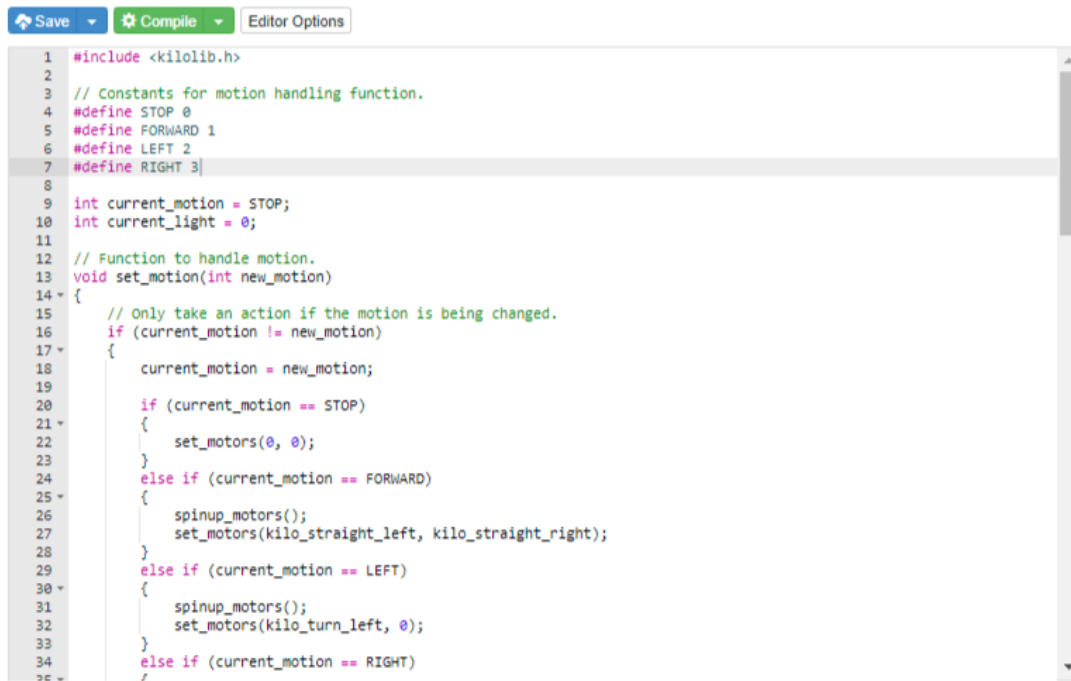
2.3 Programming Environment

We used the online editor and compiler Kilobotics [13] as the programming platform. This software system was designed by Alex Cornejo and Mike Rubenstein, and is maintained by the Harvard Self-Organizing Systems Research Group. Using this system, we can write the program on the web that is stored as a C file in a dropbox. The program is compiled online using built-in libraries there, and the generated hex file is downloaded into the dropbox directory. The programming environment consists of the following components.

Kilobot Library: The Kilobot is programmed in C. On the Kilobotics online editor, the libraries related to robotics movement and communication (controlling motors, sensors, and navigation, and so on) are all installed and accessible by default.

Editor and Compiler: The Kilobotics online editor allows us to write and compile programs for the Kilobot. It uses Amazon servers for compilation, and as a user, we can store all of our C program files and the compiled hex files in the dropbox directory. The Kilobot program

can also be compiled locally, but mostly, we have used the online compiler, which is easy to use, with debugging information inside the editor, as shown in Figure 10.



```
1 #include <kilolib.h>
2
3 // Constants for motion handling function.
4 #define STOP 0
5 #define FORWARD 1
6 #define LEFT 2
7 #define RIGHT 3]
8
9 int current_motion = STOP;
10 int current_light = 0;
11
12 // Function to handle motion.
13 void set_motion(int new_motion)
14 {
15     // Only take an action if the motion is being changed.
16     if (current_motion != new_motion)
17     {
18         current_motion = new_motion;
19
20         if (current_motion == STOP)
21         {
22             set_motors(0, 0);
23         }
24         else if (current_motion == FORWARD)
25         {
26             spinup_motors();
27             set_motors(kilo_straight_left, kilo_straight_right);
28         }
29         else if (current_motion == LEFT)
30         {
31             spinup_motors();
32             set_motors(kilo_turn_left, 0);
33         }
34         else if (current_motion == RIGHT)
35         {
```

Figure 10. Kilobotics program editor

2.4 Shape Formation

Shape formation is a popular task for swarm robots. There are two common processes of shape formation, i.e., subtractive shape formation and additive shape formation [10], [17]. In both cases, there are some advantages and disadvantages. In subtractive shape formation, a lot of robots that are not part of the desired shape finally move away. On the other hand, in the additive approach, it is likely that the shape border may be incomplete due to a random combination of the robots and the geometric form of the desired shape.

2.4.1 Subtractive Shape Formation

Subtractive shape formation is relatively simple in that the robots within the desired shape do not need to move. However, it requires a large number of robots to contain the shape

initially. First, we will gather all the robots as a group and let them figure out their location through multilateration. Then based on the desired shape, each robot will decide whether it's inside of the shape or not. If it's not a part of the shape, it will evacuate itself from its position away from the cluster where the in-shape robots remain. Thus, the shape will be formed as demonstrated in the flowchart in Figure 11.

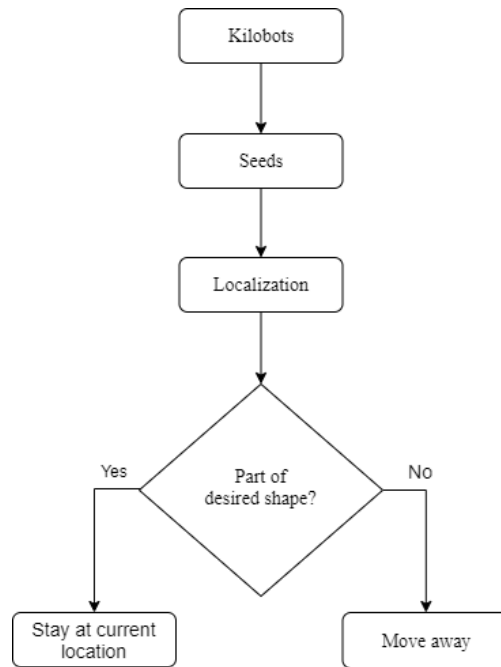


Figure 11. Subtractive shape-formation

In this process, there is a possibility that a lot of robots may not be a part of the shape and hence those robots do not contribute to the final shape. It is a limitation of this process. On the other hand, this process is fast when the robots that need to move away are guided by a light source. We need a lot of robots to perform this test in the lab, and all the robots should be of the same dimension and similar in nature to interact with each other. The previous design of WCU Kilobot version 1.1 can perform light intensity related tasks only in a dark environment. To perform in the room-light condition, the ambient light sensor circuit needs to be adjusted.

2.4.2 Additive Shape Formation

Additive shape formation is a comparatively complex action that overcomes the limitation of the subtractive shape formation, where some robots move away from the desired shape to be not used eventually. But in the additive approach, all the available robots can contribute to forming the shape as shown in Figure 12.

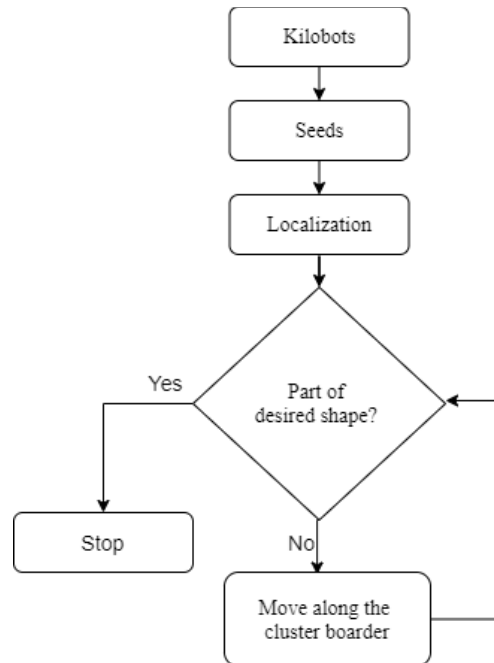


Figure 12. Additive Shape-formation

In additive shape formation, the robots move along the cluster of other robots until they stop at a position and contribute to the desired shape. This process is typically lengthy and slow. However, it complements the subtractive approach and it is beneficial to combine both approaches to achieve both speed and accuracy.

CHAPTER3: METHODOLOGY

3.1 Building and Challenges

We have built Kilobots here at WCU and experienced the building and debugging challenges. The building process was tedious when debugging was an essential step. There were still PCB boards available from the last design at WCU, Kilobot version 1.1, on 3 by 3 panels. The SMD components were no smaller than imperial 0603 size (0.06 inch by 0.03 inch, or 1.55 mm by 0.85 mm). We ordered a stencil, as shown in Figure 13, to go with the PCB board while applying the solder paste. Once we aligned the stencil and the PCB board well and taped them in place, we spread a thin layer of solder paste onto the stencil so that an appropriate amount of solder was applied to the PCB.

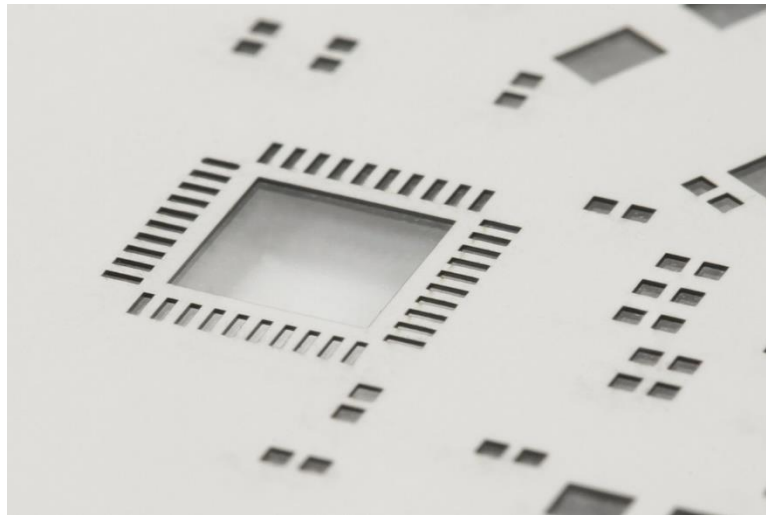


Figure 13. The Stencil that we used

Then we used a tweezer to place the components onto the board, which would stay in place due to the slight viscosity of the solder if the board was not shaken, and then the board was baked in an electronic reflow oven nearby our working bench minimizing the unintended displacement of the components before baking. After all the components were placed, we first visually examined the board with a 10x zoom lens to check if there is any unexpected short, and

we could fix those short before the board was baked in a reflow oven. The reflow oven we used is shown in Figure 14.



Figure 14. The reflow oven we used

After being baked, the circuit boards would look like Figure 15. There were typically numerous shorts as the solders would melt and spread during the heat treatment. Our debugging procedures are described in the following section.

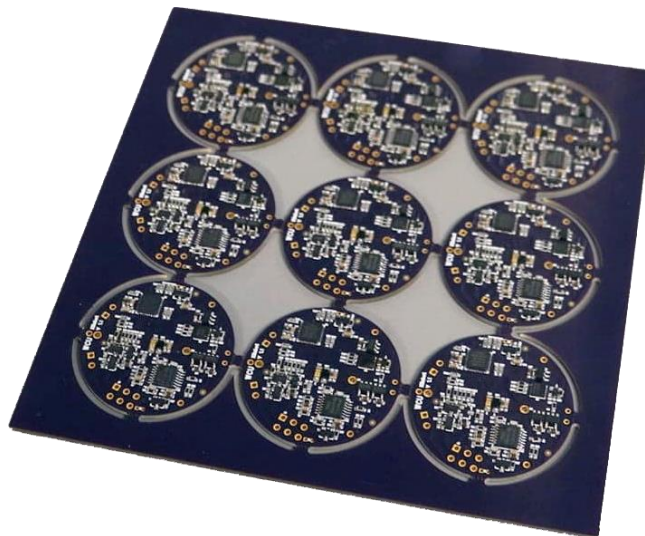


Figure 15. Kilobot board (SMD mounted)

The previous Kilobot design (WCU Kilobot version 1.1) has already replaced all components in the original Harvard design to be no smaller than 0603 packages, but soldering by hand is still a challenge to place the SMD components. The spacing between the components is

tiny, making it difficult to avoid unexpected short circuits. Some regions on the board are more crowded than the others, and those regions often need to be debugged.

To reduce unexpected shorts, we choose the SMD291AX10T5 solder paste with a mesh size of T5 where the solder particles are the smallest on the market (15-25 μm , on average). We have also ordered a new stencil with even thinner slots than before, to leave less amount of solder onto the PCB board. Both adjustments helped us to reduce unexpected shorts.

As an example of debugging such unexpected shorts, Figure 16 shows a microscopic view of the board before the debugging, where there were unexpected shorts between the leads on the bottom side of the chip. The IC chip, labeled as U6 in the schematic, is an op-amp with seven leads on two sides of it. To examine it, we used both the 10x zoom lens for eye-viewing and the pico zoom of the electronic magnifier that displayed the view on the computer monitor. Figure 17 shows the board after the debugging, where the leads at the bottom side of the chip, U6, were separated.

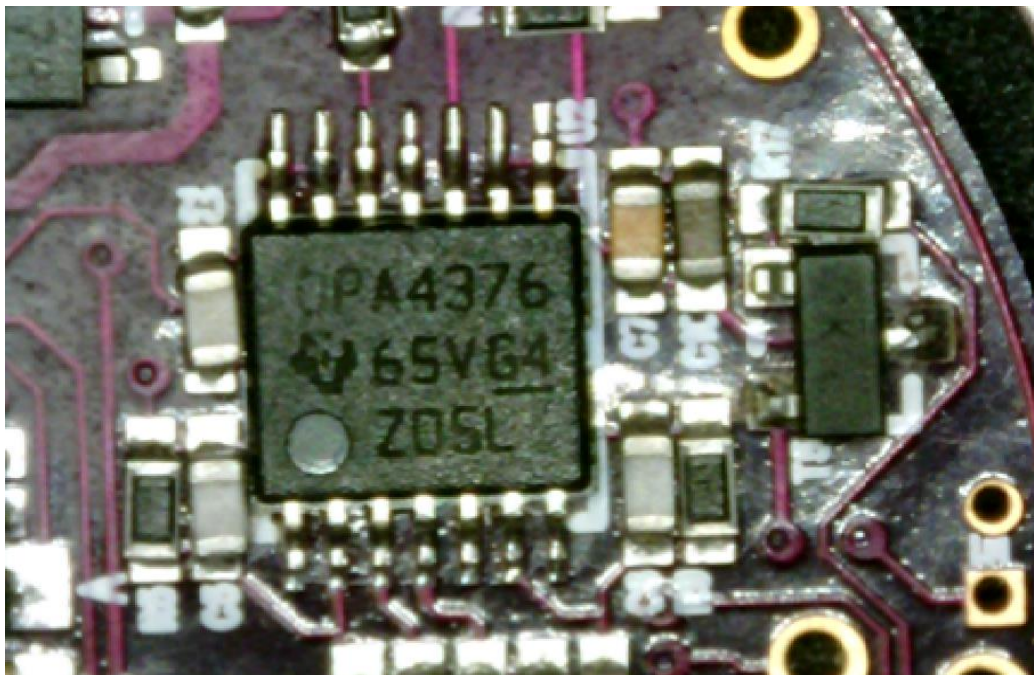


Figure 16. Kilobot board microscopic view before debugging

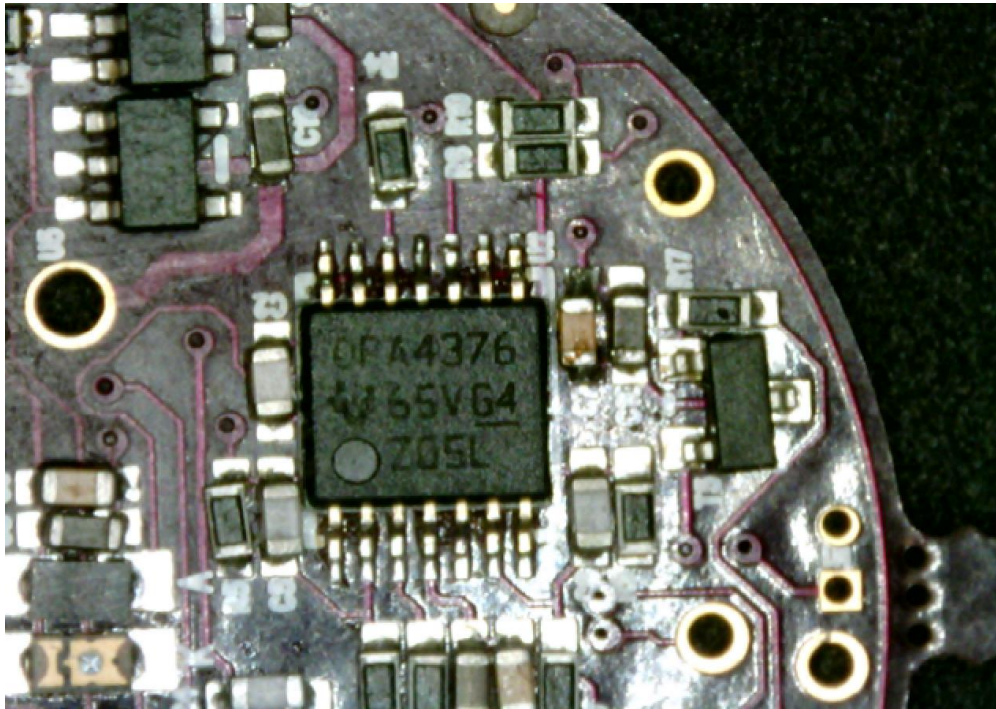


Figure 17. Kilobot board microscopic view after debugging

Even if the circuit was built carefully, it was almost impossible to avoid unexpected short circuits. So debugging was an essential step while building Kilobots. There were mainly three scenarios of unexpected shorts: (i) There was some unexpected short due to excessive solder paste, but the excess solder was visible; (ii) The short was caused by the displacement of the components so that two components came in contact with each other directly; (iii) The solder got under the chip excessively, which was not accessible nor directly debuggable, and we should not heat the chip too much to damage it.

In the first situation, we heated the board using a hot air gun and took the extra solder paste away with solder removal wick wire.

In the second situation, there was no way to get rid of short by simply removing the solder. We had to reposition the components. Therefore, we heated the components carefully and repositioned them using a tweezer. For example, Figure 18 shows the squire chip in the center

(labeled as U2 on the schematic) is surrounded by a group of three resistors on its left and a capacitor and two other components on its right. There was no adequate space between the chip and its adjacent components to debug the chip U2. We had to remove the adjacent components to fix the unexpected short of the chip beneath it and then re-mounted those surrounding components back to their original positions. The whole process was time-consuming and very inconvenient. So, to eradicate this issue in our revised design, version 1.2, we have increased the spacing between the components.

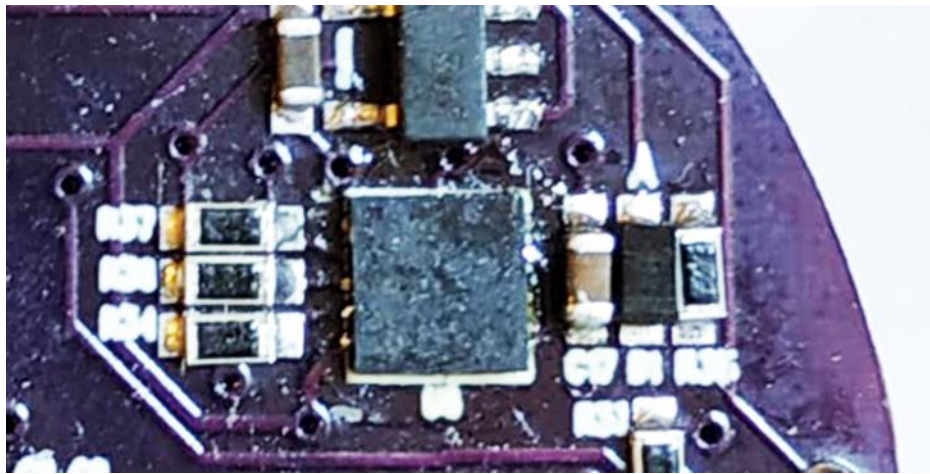


Figure 18. Kilobot board view of inaccessible parts

In the third situation, it was very tricky to find such a short and fix it, but fortunately, it was rare in my building experience. Sometimes the main microcontroller unit was shorted at its bottom, which was not visible even in a microscopic view. After thoroughly debugging a board, if I still couldn't get a reasonable impedance between the battery opening and between other measurement points, I would assume this internal short issue. I found several cases of this issue and fixed them by heating the microcontroller carefully. Most of the chips, including the microcontroller, can tolerate a heat source of 120° C from a reasonable distance for no more than 30 seconds continuously. Sometimes there was unexpected short at the bottom of the chip so I

had to heat it to loosen the solder. I would heat the chip for about 15 seconds at a temperature of 110° C and then pushed it from the top to press the excessive solder out. Then I use the solder removing wick wire to get rid of the extra solder.

3.2 Light Based Operation

Kilobots is equipped with an ambient light sensor useful for the operations based on light intensity. The ambient light sensitivity of the previous designs (Harvard and WCU version 1.1) is suitable only for simulations in a dark room. We aimed to update the Kilobot design in WCU version 1.3 to operate them in a more tolerant lighting condition than in a dark room [18].

To examine the ambient light sensor, we built an equivalent circuit using Arduino Uno and the same kind of phototransistor used in Kilobot. A suite of emitter resistance values (from the existing design's 608 k Ω down to 10 k Ω) was used to measure the output voltage at various illumination lux levels, as shown in Figure 19. The exact resistor values were measured and recorded in Table 2, and then the sensor output at various lux values was measured and reported in Table 2. The same data were plotted in Figure 20 for easy visualization.

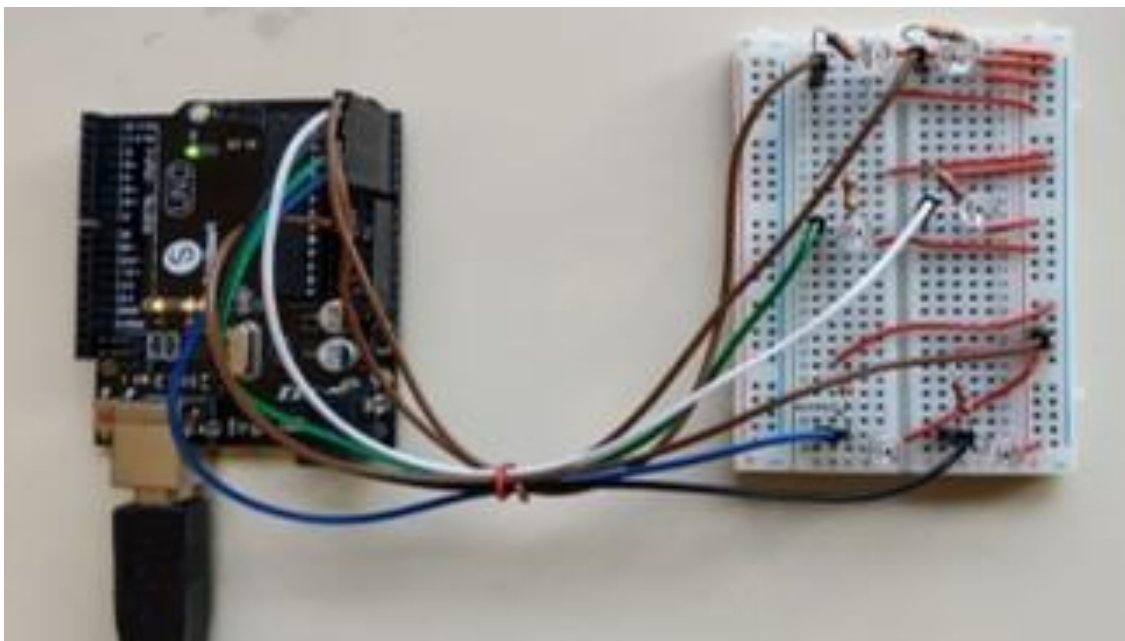


Figure 19. Ambient light circuit using Arduino Uno

Table 2. Ambient light sensing using the Arduino Uno

Intensity(lux)	2	40	90	200	350	500	600	817	1000	1504	2364	3278
$R_{35}=9.70\text{ K}\Omega$	4	14	86	139	181	210	360	369	420	560	618	844
$R_{35}=26.09$	8	38	190	359	444	564	670	954	962	976	980	982
$R_{35}=35.34$	27	252	466	528	580	680	814	974	980	982	986	992
$R_{35}=60.25$	43	184	901	960	978	968	999	1001	1003	1006	1007	1012
$R_{35}=208.4$	60	1003	1004	1002	1006	1003	1005	1005	1007	1008	1012	1015
$R_{35}=598.0$	105	1009	1006	990	992	993	994	996	998	998	1000	1017

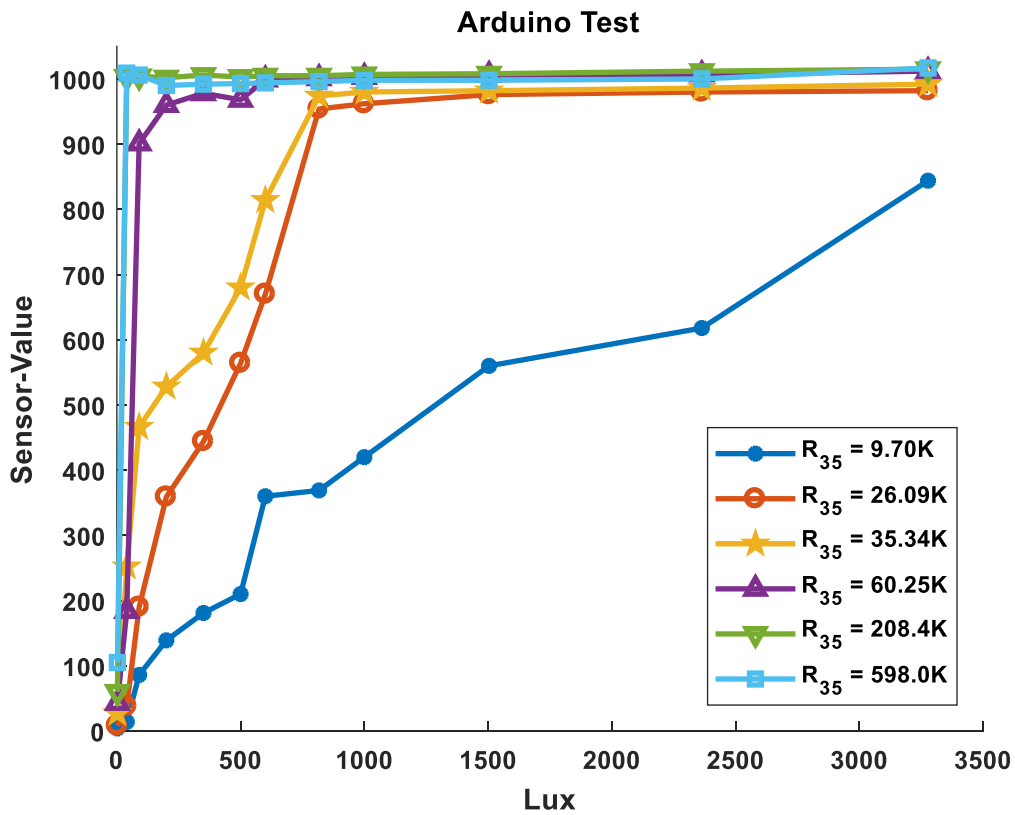


Figure 20. Sensor reading vs. light intensity using Arduino Uno

Next, we replaced this particular resistor on Kilobot, labeled as R_{35} in the schematic, by a potentiometer soldered onto the board, and tried the same experiment using the serial channel of the KiloGUI, as shown in Figure 21.



Figure 21. Kilobot with a potentiometer for ambient light test

We tuned the potentiometer to be exactly the same resistance as what we used in the Arduino test. The result was reported in Table 3 and plotted in Figure 22. We found that both the Arduino and the Kilobot experiments were consistent to show that the smallest resistance in our test is the best to achieve decent sensitivity and range. If this resistance was set to be even smaller, we would lose out on the output range and incur too much current in this phototransistor circuit, which was not desirable, either.

Table 3. Ambient light sensing using the potentiometer of a Kilobot

Intensity(lux)	2	40	90	200	350	500	600	817	1000	1504	2364	3278
R₃₅=9.70	6	31	137	142	187	213	363	367	521	577	906	994
R₃₅=26.09	7	65	249	476	540	721	807	923	947	995	1000	1006
R₃₅=35.34	9	400	570	678	725	841	909	953	984	999	1004	1008
R₃₅=60.25	27	940	970	990	995	998	997	1004	1006	1008	1011	1018
R₃₅=208.4	72	1007	1009	1011	1012	1013	1013	1014	1016	1017	1022	1023
R₃₅=598.0	115	1009	1011	1012	1014	1015	1015	1017	1018	1020	1022	1023

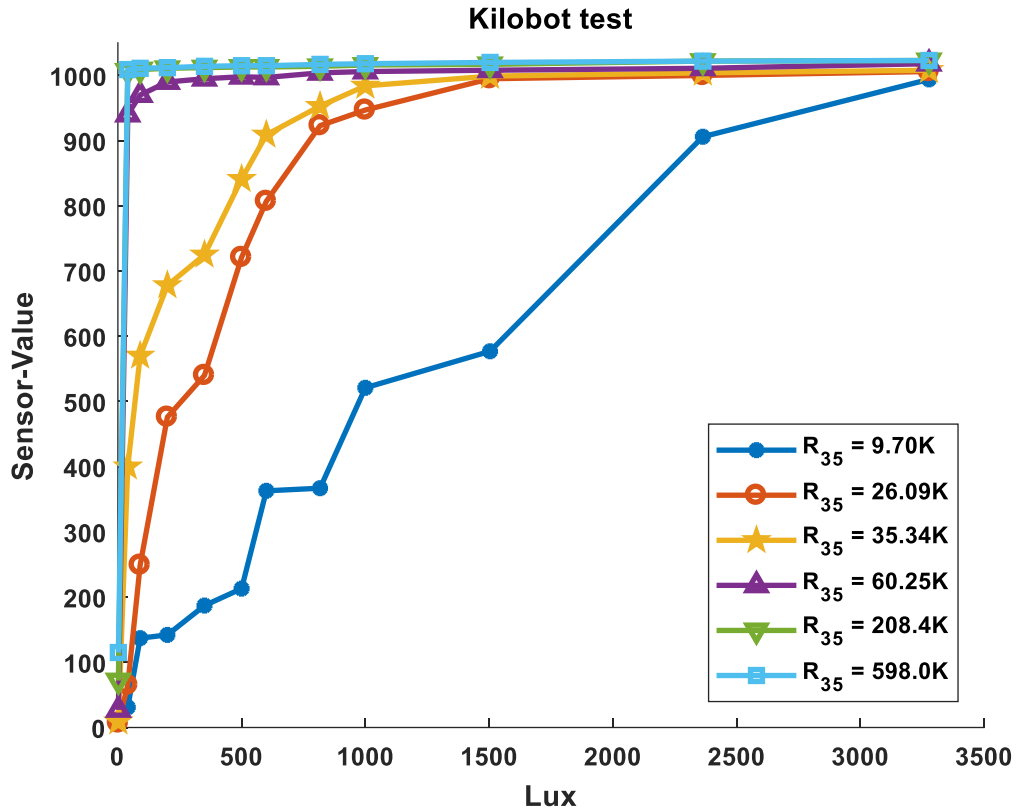


Figure 22. Sensor reading vs. light intensity using modified Kilobot

As shown in Figure 22, when the emitter resistance is at 598 k Ω (close to the previous design 604 k Ω), the response saturates at a low illumination level of about 90 lux. However, the regular daylight in-door is measured to be about 1000 lux in the Lux Light Meter app. We found that the resistance value of $R_{35} = 9.7 \text{ k}\Omega$ gave the widest response. Our BOM has already included an 11 k Ω resistor, so we replaced the resistor $R_{35} = 608 \text{ k}\Omega$ by 11 k Ω (given the trend of the data, it is expected that 11 k Ω will behave similarly as 10 k Ω).

3.3 Insufficient Memory Issue

ATmega328P is a high-performance Microchip picoPower 8-bit AVR RISC-based microcontroller combined with 32KB ISP flash memory. A portion of this memory (about 5-6 KB) is used for the bootloader file. In a large-scale complex operation, the loaded program may exceed the amount of remaining memory on the chip causing real-time malfunction. Efficient

memory usage in coding is needed. Meanwhile, we were considering other microcontrollers that could replace the ATmega328P microcontroller with bigger memory. There was no alternative to ATmega328P that provides more than 32KB programable memory if every other specification is unchanged. To incur minimum change while enlarging the memory, we especially considered the programable flash memory, the number of leads, and the physical dimension of the chip. We first considered ATmega2560 QFN, in the same family as ATmega328p. However, ATmega2560 has 100 leads, much more than the 32 leads of ATmega328P, which complicates the building process, and it is hard to fit it on the board. A careful review leads to two alternatives to ATmega328P, i.e., ATmega-1284 and ATmega-644. Table 4 shows the property comparison between the original and the two alternatives.

Table 4. Microcontroller property comparison

Property	Atmega-328P	Atmega-1284	Atmega-644
Leads	32	44	44
Flash Memory	32 Kbytes	128 Kbytes	64 Kbytes
EEPROM	1000 bytes	4096 bytes	2000 bytes
Operating voltage	1.8 to 5.5V	1.8 to 5.5 V	1.8 to 6.0 V
Temperature Range	-40°C to 85°C	-55° C to +125° C	-40° C to 85° C
Unit Price	\$2.08	\$3.46	\$3.96
Package	32-VQFN (5x5)	44-VQFN (7x7)	44-VQFN (7x7)
Core Size	8 Bit	8 Bit	8 Bit
Package Type	QFN(5x5)	44-pad VQFN/QFN/MLF	44-pad QFN/MLF
Write/Erase Cycles	10,000	10000	10000
PWM Channels	6	8	6
Active Mode	0.2 mA	0.4 mA	240 μ A
Power-down Mode	0.1 μ A	0.1 μ A	0.1 μ A
Power-save Mode	0.75 μ A (Including 32kHz RTC)	0.6 μ A (Including 32kHz RTC)	N/A

For all the alternative chip replacement, the libraries for programming will likely need to be updated for the specific chip.

3.4 Debugging Model

After a board is built, there are often numerous unexpected short circuits on the board. The most common problematic regions on the board are the crowded regions, as well as the leads of the chips, such as the ATmega328P chip (in a Quad Flat No-Lead VQFN package with 32 leads), as there are often solder bridges between the adjacent leads creating unexpected shorts. A thinner-slot stencil was used to help control the amount of solder in use, but such shorts are unavoidable. Moreover, the issues on the board might be discovered at various stages of testing, but not all at the beginning. After all the parts, including the through-hole components, are soldered on, the board cannot be heated in the oven again, making debugging even more challenging.

The debugging procedures include the following:

- Use a meter with needle leads to identify any unexpected shorts, apply rosin flux, and then heat the location using the soldering iron. Extract extra solder by the tip of soldering iron or soldering wick if the extra solder is excessive.
- Use an eyepiece optical magnifying glass or a table-top magnifying monitor to identify the misaligned chips, apply rosin flux, and then use stifling air station (hot gun, at a temperature of 350 Celsius degrees and zero airflows) to heat up the chip and reposition it using a tweezer.
- The bottom of the chip is not accessible. If a bridge persists even after the rework, heat the chip using the hot air gun with controlled duration and temperature so as not to burn the chip, and then press the chip using a tweezer to push the extra solder out to fix the bridge.

- Meanwhile, there can be also unexpected open on the board. We will first do a visual inspection under a microscope, and then test the impedance between pairs of contact points (such positions are listed in a spreadsheet, and the impedance between all those testing points on that list will be examined) to make sure that values are within a normal range. If not, we will apply the solder to fix the open.

3.5 Kilobot Version Update

WCU Kilobot version 1.1 was designed earlier by the WCU Kilobots team [19], [20]. Based on our building and debugging experience, we made a few revised designs. The WCU Kilobot version 1.1 was designed in PADS PCB Design Software by Mentor Graphics. The PADS PCB Design license was no longer available, and we used Altium Designer to repopulate the earlier design based on schematics, and then redo the layout in a similar fashion. With the source files of the schematic and layout in Altium, we could then make revisions to the Kilobot design, as shown in Figure 23. The detailed steps we took are explained below.

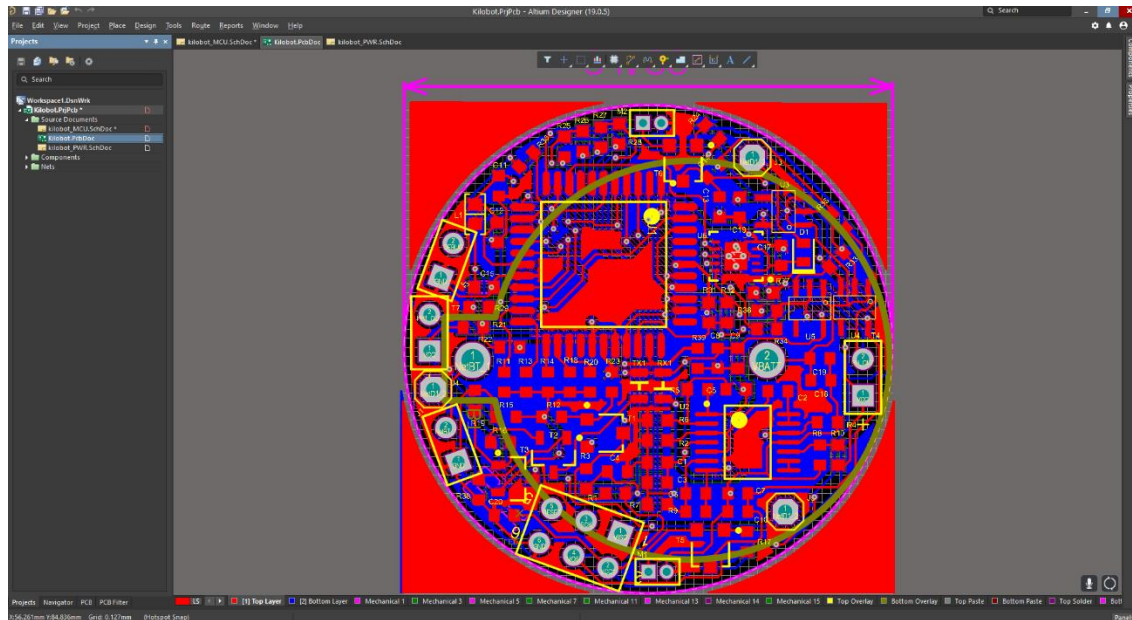


Figure 23. Workspace in Altium Designer

(1) Creating the schematics:

First, we created a PCB project inside the Altium Designer. Then from the “add new project” option, we added two blank schematics (one for the microcontroller unit and one for the power unit, in SchDoc) files and one blank PCB (in PcbDoc) file for our design. The schematics show which components are used, how they are connected, and the relationship between the groups of components. In order to keep things organized, we have divided the schematics into two sections, i.e., the microcontroller unit and the power unit. We installed libraries according to our project. In addition, we built libraries for some components by drawing schematics and PCB footprint. Then from libraries, we clicked and dragged the components on the schematic sheet and positioned them at a convenient place. Sometimes, we changed some properties of components for our needs. Eventually, we completed the wiring of the components, and Altium annotated all the components we used.

(2) Initializing the PCB:

After completing the schematic, we switched to the blank PCB layout (PcbDoc file) that we created at the beginning of the project to start the PCB design. After that, from view configurator, we configured the view of our PCB such as the color of the top and the bottom layers. We also set a polar and a cartesian grid to align our components nicely. Then, we defined the board shape as a circular board.

(3) Designing the PCB Stack-up:

Before transferring our schematic information to the PcbDoc file, we configured the layers of the board. In order to modify the layers, we used the “Layer Stack Manager” option from the “Design” drop-down menu. For our project, we defined a double-layers PCB in Altium Designer that includes the top layer and the bottom layer. We also saved this stack manager setting as a template for future use.

(4) Pushing schematics footprints from schematic to PCB:

Unlike other PCB design software, Altium Designer works in a synchronous design environment. So we were able to access the schematic design, PCB layout, and the BOM file simultaneously. After a successful compilation of the schematic design, the information in the SchDoc file was ready to be imported into the PcbDoc file. Schematics can be imported into the PCB in two ways: (i) If working with the schematic, from the Design menu, click Update PCB Documents. (ii) If working with the PCB, use the Import Changes feature under the Design menu. As we made some changes in the schematic while completing the PCB, we used either of the options to keep our PCB design synchronized with the schematic file.

(5) Placing Components:

Altium Designer provided some flexibility to quickly place components on the circuit board. First, we made an automatic placement for the components and then made some major modifications of the placement to make the components organized according to our design. “Arrange components as groups” feature gave us some extra facility to place the components as a group. Also, the shortcut for switching between layers and rotating components was a great help to place all the components easily.

(6) Defining Design Rules:

Design rules check is an important feature to use for PCB design verification. We used a “PCB rules and violations” feature, which checks for 31 standard PCB design rules, to check any potential issues, warnings, or errors. We went through each of the rules and fixed any potential errors found. While selecting the rule, we need to take into account the tolerance of our preferred PCB manufacturer.

(7) Inserting Drill Holes:

Before routing the traces, we needed to insert drill holes (mounting and vias) in our design. A via consists of two pads in corresponding positions on different layers of the board, that are electrically connected by a hole through the board. We modified some of the via locations during trace routing from the via properties dialog window. We also followed the guideline by the design for manufacturing (DFM) specifications of standard PCB manufacturers.

(8) Routing Traces:

After placing all the components, we routed the traces. We have followed the recommended routing guidelines by taking advantage of Altium Designer tools to simplify the process, such as highlighting nets and color-coding via routing. Altium provided a few powerful autoroute tools that helped us to make routing comparatively easy and productive.

(9) Adding Labels and Identifiers:

Finally, after having verified the circuit board, we added labels, identifiers, and markings to the board. We referenced identifiers for all the components that would help us during the PCB assembly. Meanwhile, we carefully placed polarity indicators on the design that will help identify components and their orientations.

(10) Generating the Design Files:

We have verified our circuit board layouts by running a design rule check (DRC). Though Altium Designer had an automatic check for the layout of our components, we had multiple runs of DRC manually to ensure its reliability. Then it is ready to generate the design files to be sent to the PCB board manufacturer. We will generate a Gerber file or any other CAD file based on the requirement of the manufacturer.

CHAPTER 4: RESULTS

4.1 The first revised design (version 1.2)

The most time-consuming issue in building the Kilobot was to fix the unexpected short circuit. Figure 24 shows the WCU Kilobot version 1.1 PCB layout, where the crowded regions that often had shorts are highlighted. We have revised the design to distribute the components with wider spacing (0.15 mm in the crowded areas) than the last design (0.1 mm in the same area).

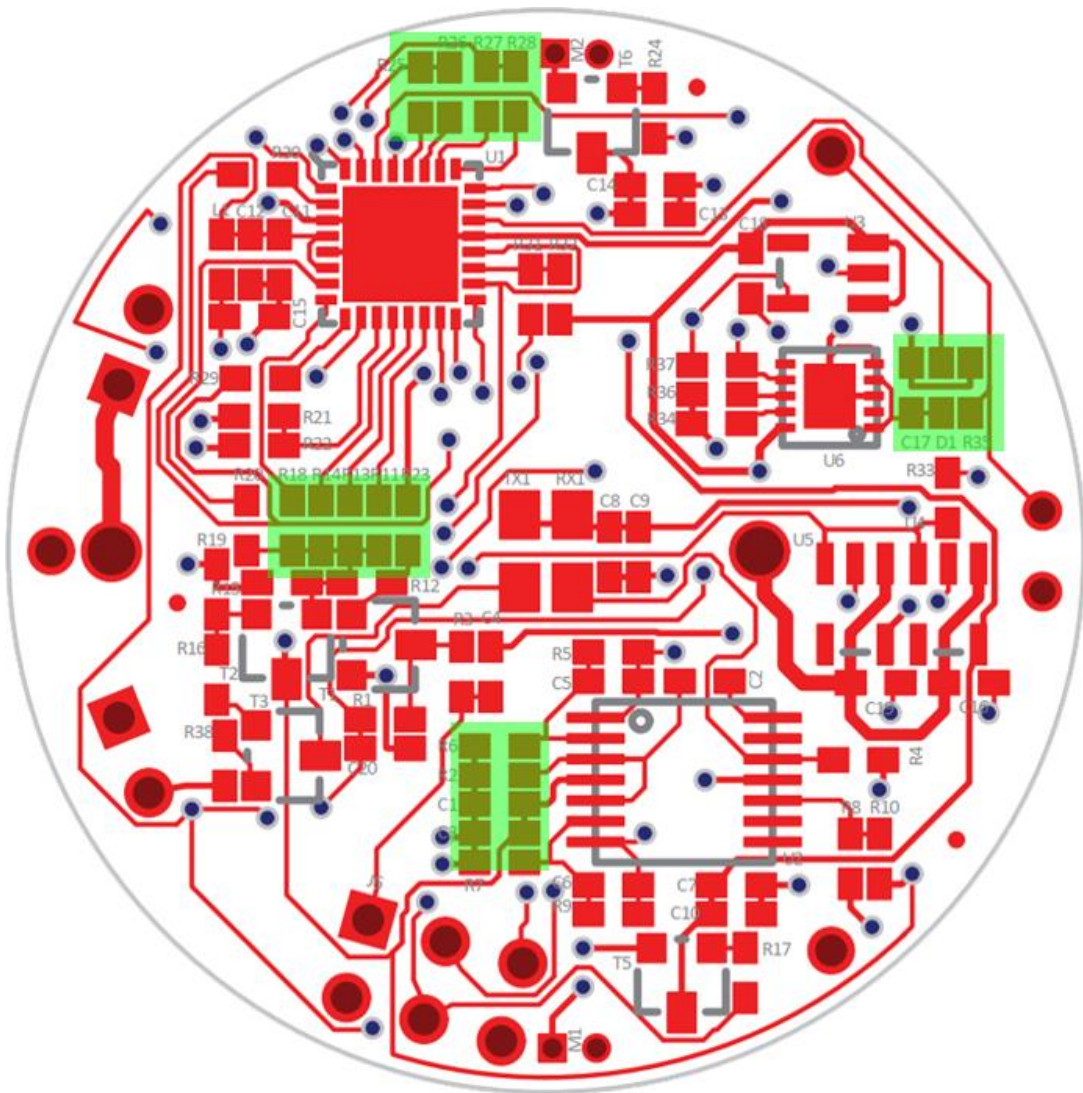


Figure 24. Schematic of Kilobot version 1.1

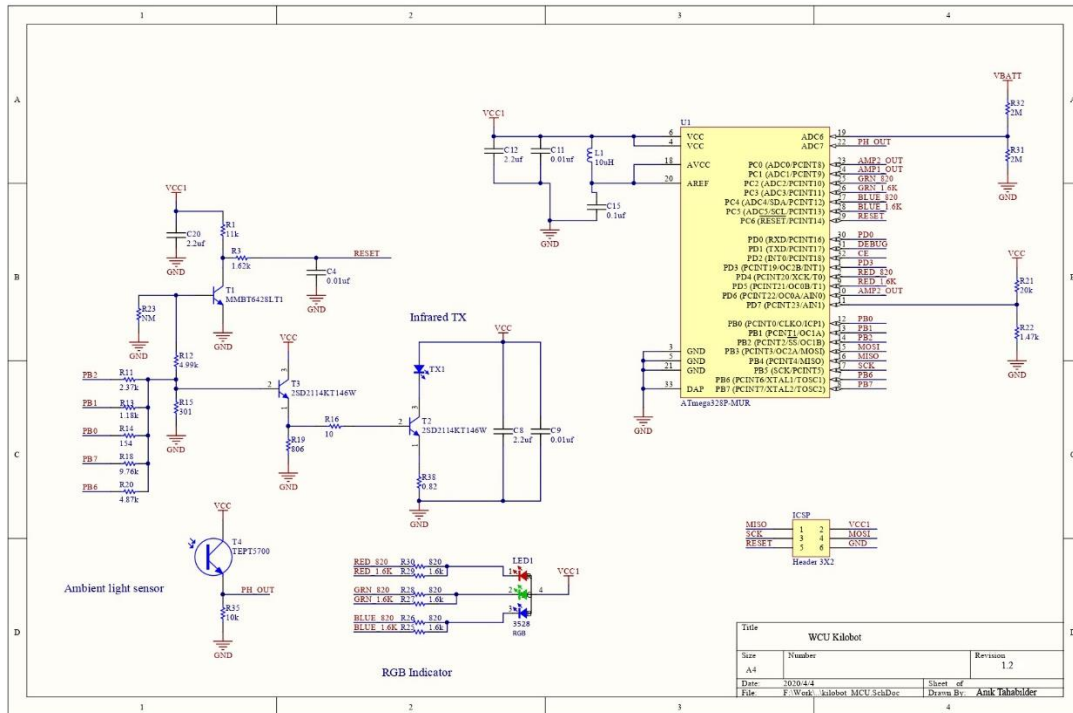


Figure 25. Schematic of Kilobot version 1.2 (Microcontroller Unit)

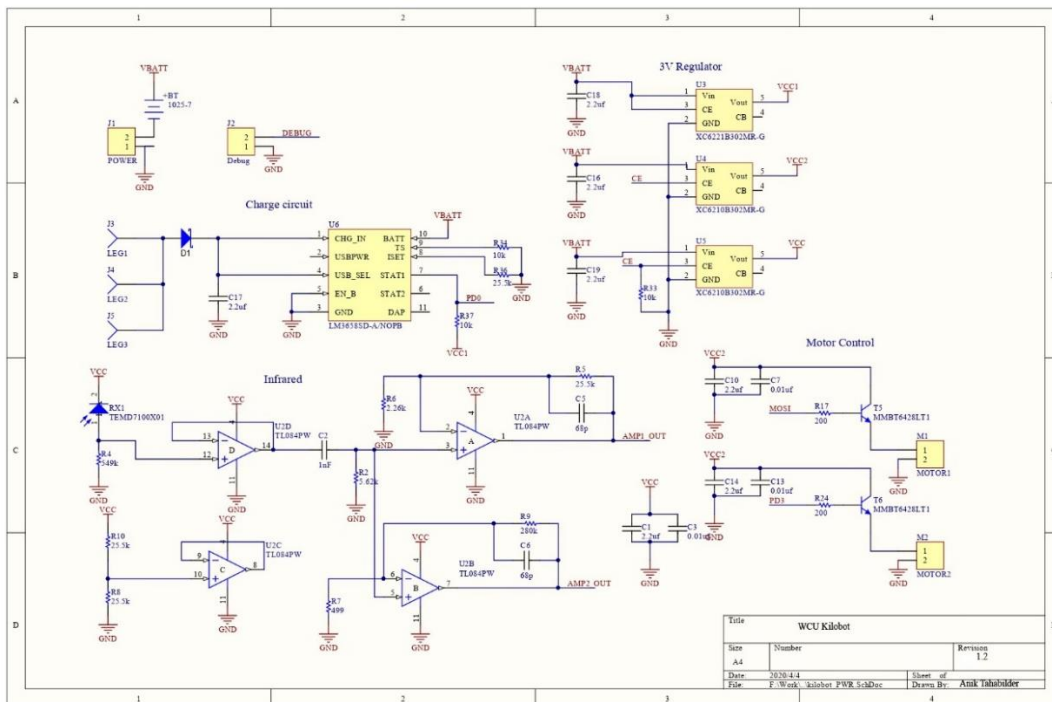


Figure 26. Schematic of Kilobot version 1.2 (Power Unit)

The WCU Kilobot version 1.2 (with its schematics shown in Figure 25 and Figure 26, and its PCB layout in Figure 27) allows us to make further modifications easily as Altium Designer is available at WCU. Version 1.2 is a two-layer PCB design, in which the components in red are on the top layer, and the wiring in blue is the bottom layer. The pads are shown in gray, also in the top layer. In Figure 27, we have highlighted the sections that are modified to make the spacing bigger than that in version 1.1.

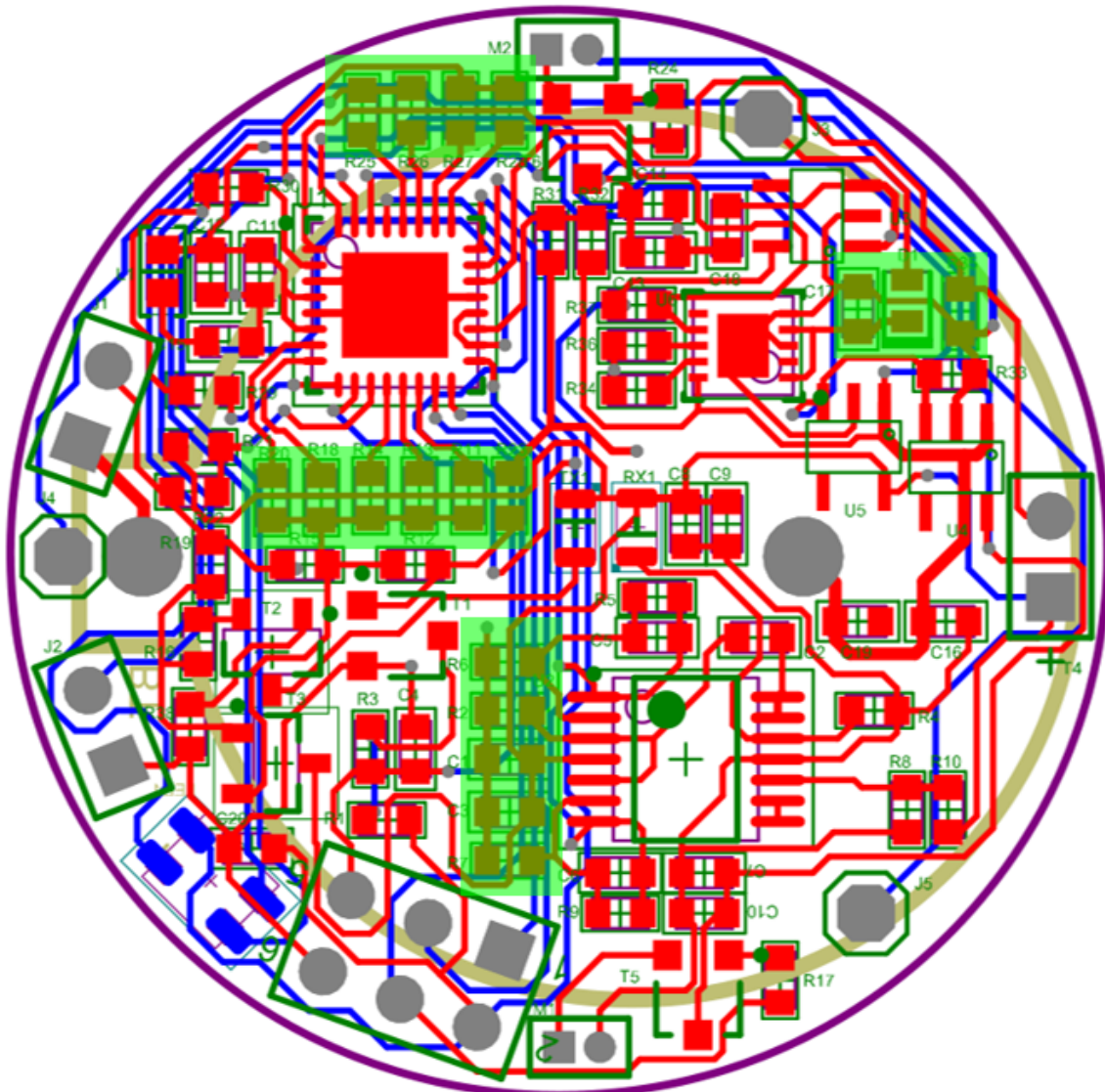


Figure 27. PCB layout of Kilobot version 1.2

After completing the PCB layout, we exported a 3D view of the PCB design as shown in Figure 28 to make an easy visualization of the components. For clarification, the top layer of the PCB board is where the SMD will be soldered on, but it is the bottom side of the Kilobot, and the Kilobot will carry the battery, ambient light sensor, and the jumper wire socket, etc. on its top. The three sticks visible in Figure 28 are the legs of the Kilobot: two are powered by vibrational motors to allow the Kilobot to move in the slip-stick fashion, while the third leg is for support.



Figure 28. 3D view of PCB layout of Kilobot version 1.2

4.2 The second revised design (version 1.3)

In this section, we focused specifically on the light driven movement. The schematic of the ambient light sensor is shown in Figure 29. Although the phototransistor circuit is simple and straightforward, the choice of R_{35} impacts the sensitivity of the sensor, as well as in what lighting conditions the sensor work.

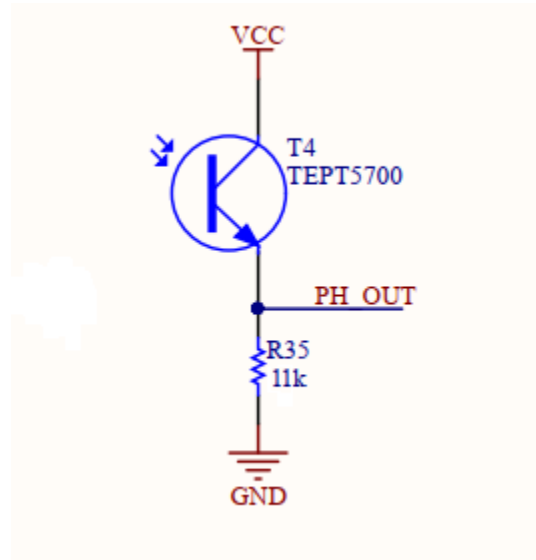


Figure 29. Ambient Light Circuit update in second revised (WCU Kilobot version 1.3)

After the experiments using both an Arduino circuit and a Kilobot with a potentiometer as R_{35} , as described in section 3.2, we adopted $R_{35}=11\text{ K}\Omega$ in version 1.3 and built a few Kilobots using $R_{35}=11\text{ K}\Omega$. We compared the Kilobot in version 1.1 and version 1.3 under several lighting conditions to carry out the phototaxis movements.

Move away from light

The lighting condition to conduct the experiment environment was created by using a single source light from the top. The light intensity at the center of the circle is very high and gradually decreasing towards the perimeter. The old Kilobot version 1.1 would not perform the move-away-from-light well unless the illumination level is low. But the Kilobot version 1.3

updated resistance value of $R_{35}=11\text{ K}\Omega$ to have a wider range of ambient light sensitivity and performed well in a wider illumination range of 500 lux to 8000 lux.

The version 1.1 and version 1.3 Kilobots were compared in the moving-away-from-light experiment at the regular daylight, out of 5 trials at each distance and orientation relative to the center of the light source. The two locations are at 12 and 20 cm away, respectively, from the center of the light source. The four orientations include facing the light (at 0 degrees so that the robot needs to turn first before it can move away), with the light on the right (at 90 degrees), with the light on the left (at -90 degrees) and with the robot backside facing the light (at 180 degrees, so the robot can move directly forward if it is at its best judgment). The counting of successful runs in the experiment is listed in Tables 5 and 6:

Table 5. Successful move-away-from-light test result using $R_{35}=604\text{ K}\Omega$ out of 5 trials

Angle (degree)	Distance from the center (cm)	
	12	20
0	0	0
90	0	1
-90	0	0
180	0	1

Table 6. Successful move-away-from-light test result using $R_{35}=11\text{ K}\Omega$ out of 5 trials

Angle (degree)	Distance from the center (cm)	
	12	20
0	5	5
90	5	5
-90	5	5
180	5	5

The 0-degree angle is the most challenging situation out of the four orientations, and an example at this orientation by each of the version 1.1 and version 1.3 Kilobots are presented.

Figure 30 shows the starting point of an old Kilobot (version 1.1) that started to move towards the high intensity of light (the green LED light on the Kilobot indicated that it was moving forward), and then the sensor read the intensity of light. The output value became saturated and the Kilobot stopped (the white LED light on the Kilobot indicated that it had stopped), as shown in Figure 31.



Figure 30. Move away from light test using old Kilobot (Starting)



Figure 31. Move away from light test using old Kilobot (Stopped due to saturation)

We conducted the same test using the new Kilobot (version 1.3) in the same lighting condition. Kilobot first read the light intensity and started to move forward, as shown in Figure 32. After moving forward for a small distance, it read the light intensity and the intensity being greater than the earlier reading, so it started to rotate left (the red LED light on the Kilobot

indicated the left rotation) to reverse direction as shown in Figure 33. Then it started to move forward towards the direction of the darker area, as shown in Figure 34.

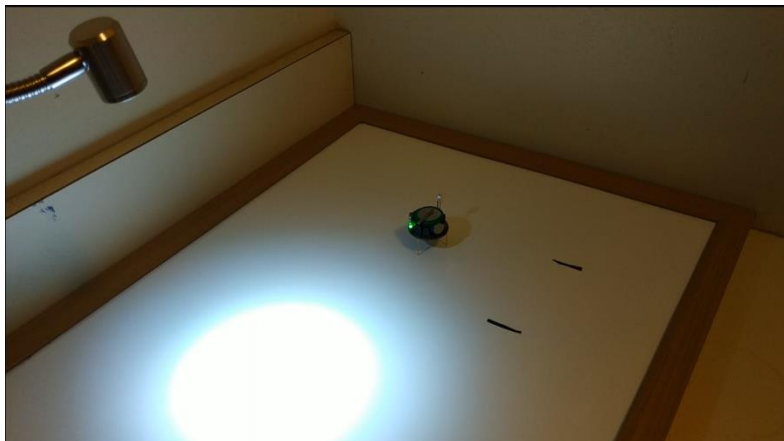


Figure 32. Move away from light test using new Kilobot (Starting)

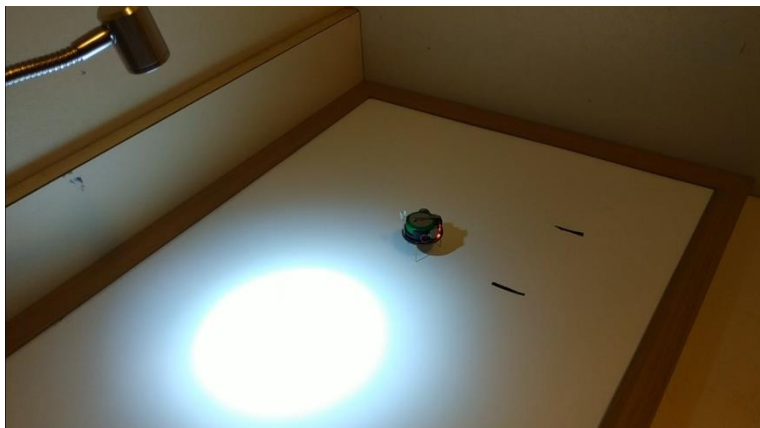


Figure 33. Move away from light test using new Kilobot (Turning)



Figure 34. Move away from light test using new Kilobot (Moving Away)

By completing the three steps mentioned above, new Kilobot successfully completed the move away from light operation. Note that the lux value was measured by an Android mobile phone app called Lux Light Meter [21]. A screen copy is shown in Figure 35.

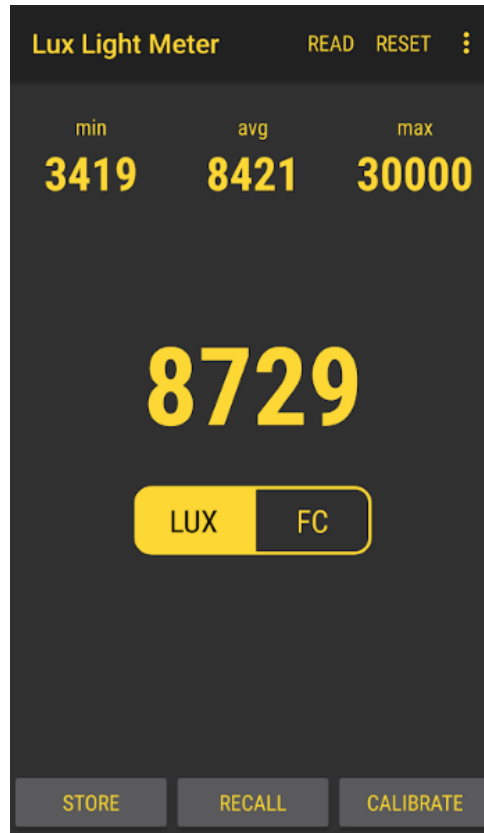


Figure 35. Measuring light intensity using Lux Light Meter

While conducting the phototaxis movement, it was challenging for the Kilobot to identify its own orientation relative to the light source. This motivated us to consider adding a second ambient light sensor at the opposite side of the existing one on the next revised version 1.4.

4.3 The third revised design (version 1.4)

To address the issue of limited memory and to enhance the controllability of Kilobots in the light-driven movements, we proposed the third revision (WCU Kilobot version 1.4) with a new microcontroller chip and a second ambient light sensor.

To examine the necessity of having one more ambient light sensor, we measured the light intensity output at a small distance increment that is very close to the diameter of the robot. We marked the distance on the table surface as shown in Figure 36. We moved the Kilobot along the perpendicular line of the circles and recorded the light intensity readings using serial communication through the overhead controller.



Figure 36. Small distance light sensitivity test for adding the second sensor

We repeated the experiment for both the old (version 1.1) and the new (version 1.3) Kilobots with the responses listed in Table 7. The new emitter resistor value in the WCU Kilobot version 1.3 showed a more significant difference between the readings than version 1.1 at a small increment of distance.

After the test result proved that two ambient light sensors at the opposite side of the Kilobot would provide quite different readings, to help it determine its orientation relative to the light source, we added a second ambient light sensor in version 1.4. The schematic is shown in Figure 37.

Table 7. Ambient light sensitivity for small distance increment

Distance (mm)	Ambient Light Sensor Reading	
	R ₃₅ =604 kΩ	R ₃₅ =11 kΩ
0	994	440
40	992	380
80	986	295
120	966	155
160	778	80
200	540	53
240	340	34
280	190	19
320	130	11
360	90	7

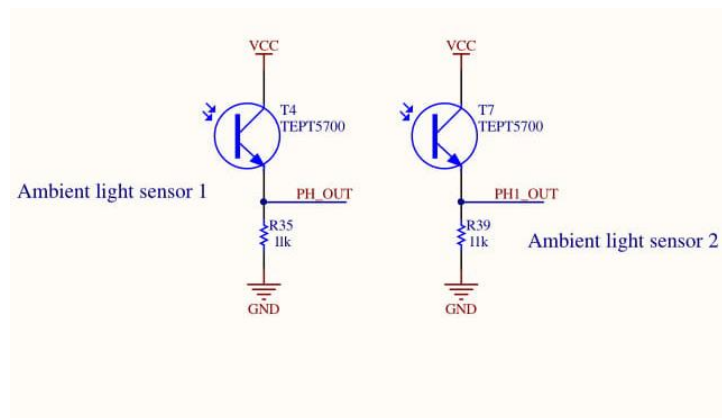


Figure 37. Schematic of two ambient light sensors

Another revision in version 1.4 was to swap out the original microcontroller by an alternative microcontroller with more memory. We decided to use ATmega1284 as the new microcontroller after the comparison in Section 3.3 Insufficient Memory Issue. Table 8 lists the key properties of the ATmega1284 chip.

Meanwhile, given the 44 leads in ATmega1284 as the IO channels (12 more than ATmega328P), the second ambient light sensor's reading can be read and processed by the microcontroller as well. After reviewing the compatibility of the new microcontroller with the

existing components on both schematic and PCB layout, we built the new schematic for the new design. The schematic of the new microcontroller chip ATmega1284 is shown in Figure 38.

Table 8. Key properties of ATmega1284

Property	Value
Flash Memory	128 Kbytes
Leads	44
Package	44-VQFN (7x7)
Core Size	8 Bit
Operating voltage	1.8 to 5.5 V

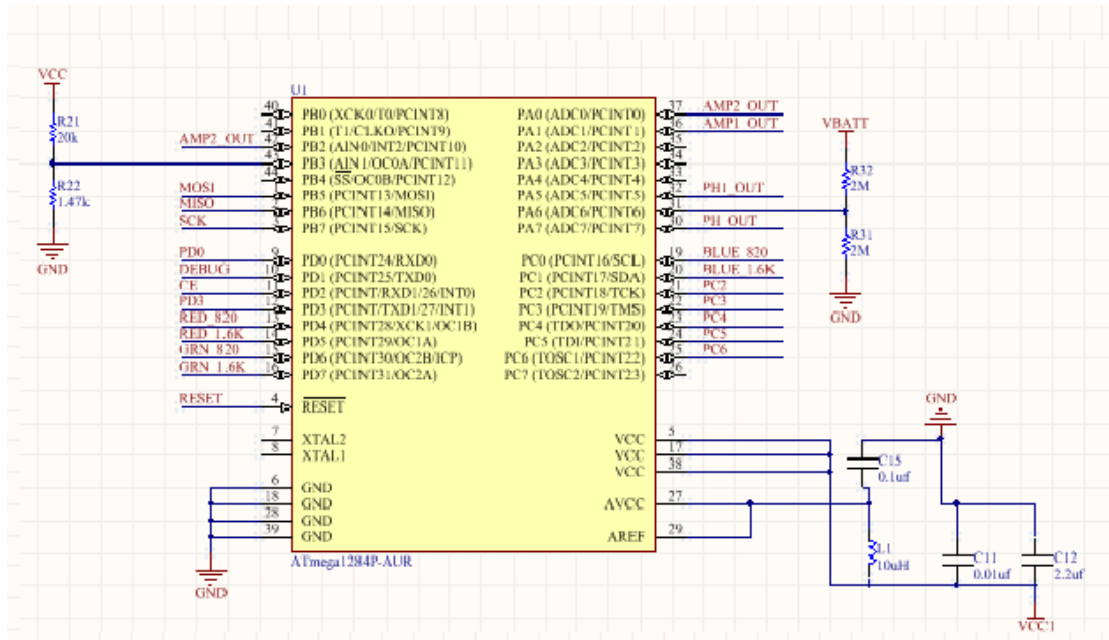


Figure 38. Schematic of the new MCU ATmega1284

After finishing the new schematic of ATmega1284, we continued to incorporate it into the full design. The overall schematic is divided into two parts as before; one is the microcontroller unit, and the other is the power unit. This microcontroller unit included one more ambient light sensor and an updated microcontroller, ATmega1284, as shown in Figure 39. The schematic of the power unit in version 1.4 is unchanged from before, as shown in Figure 40.

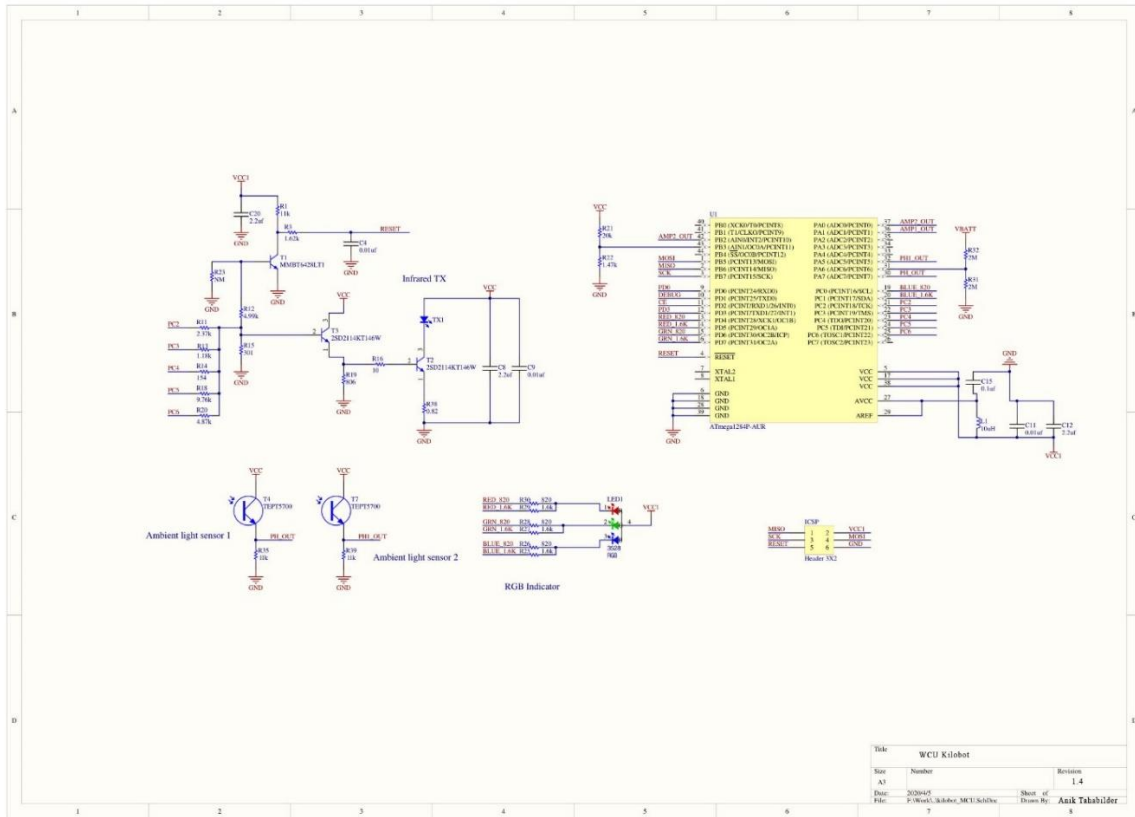


Figure 39. Schematic of the of Kilobot version 1.4 (Microcontroller Unit)

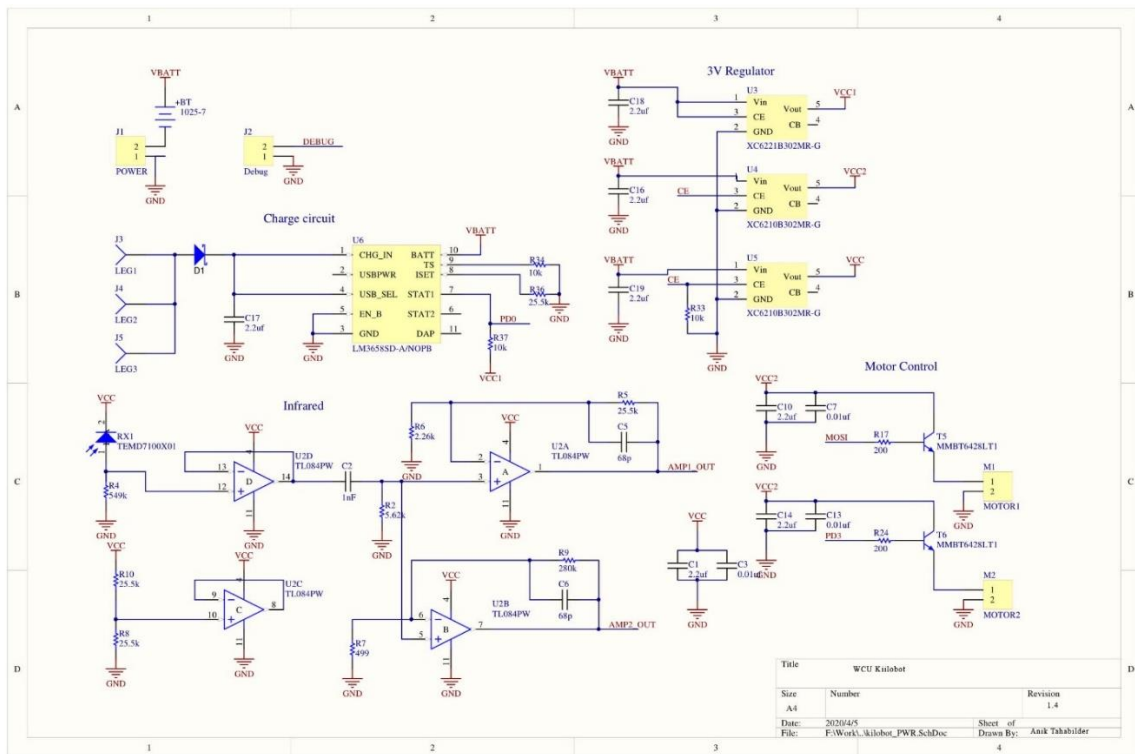


Figure 40. Schematic of the of Kilobot version 1.4 (Power Unit)

Once the schematic of version 1.4 was finished, we revised the PCB design. There are some differences in this PCB layout from the earlier versions. First, there is one more ambient light sensor. With the ambient light reading from two different sensors on the opposite sites of the Kilobot, it will improve the controllability of the Kilobots in light intensity based movement. Second, the microcontroller is replaced. Third, the components have been repositioned with enough spacing in between them (0.12 mm). Figure 41 shows the PCB design of the WCU Kilobot version 1.4.

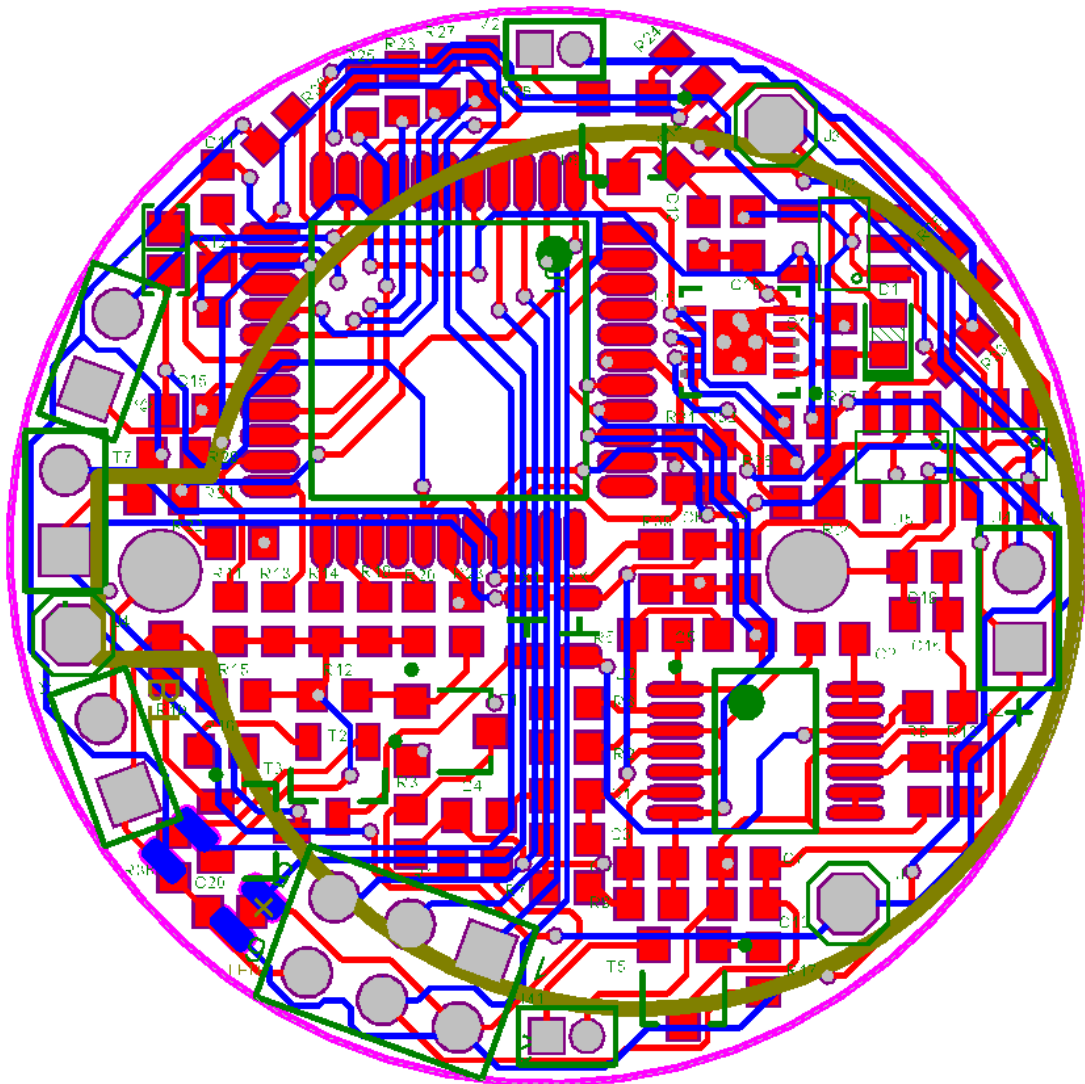


Figure 41. PCB layout of Kilobot version 1.4

Figure 42 shows the 3D view of the PCB design in the third revision (version 1.4). In the figure, we indicated the sensor position with two black arrows. The two different ambient light sensors at the opposite sides of the Kilobot will give different sensor readings at different orientations and thus help it to respond to the light easily. This design still fits on a 33 mm diameter PCB board, so it still fits the charging and storage box and is comparable with earlier designs.

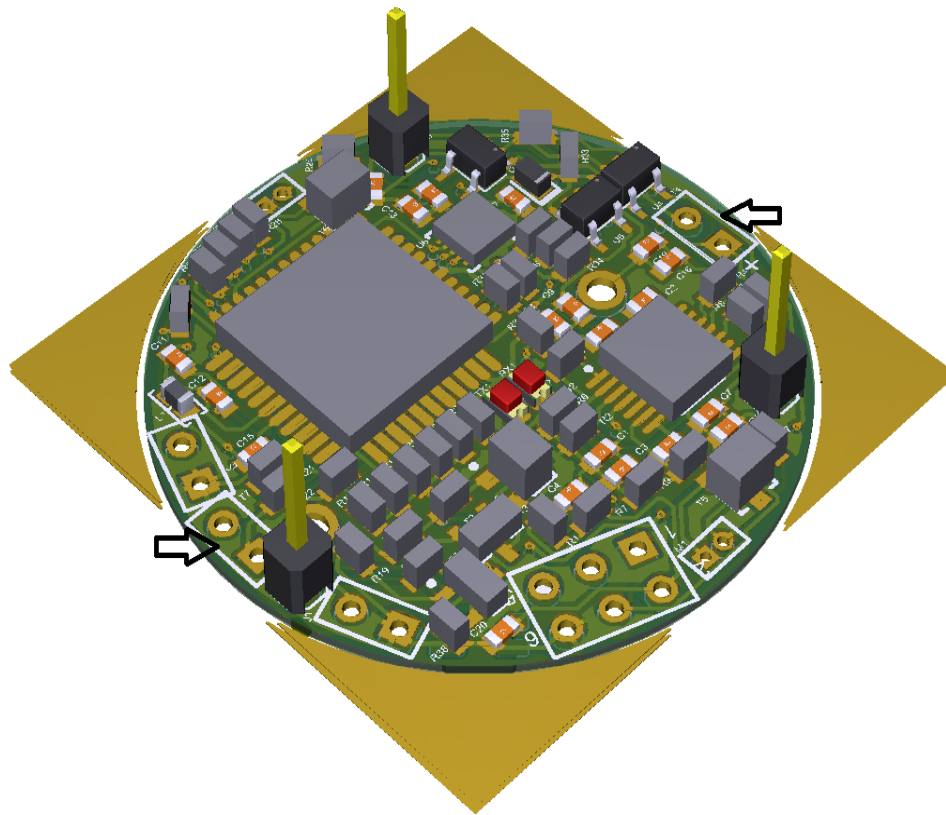


Figure 42. 3D view of Kilobot version 1.4

CHAPTER 5: CONCLUSION

There are several platforms to conduct swarm robotics simulations and experiments. Kilobot, a small robot with a 33 mm diameter, is one of the widely used platforms to test swarm action in a multi-robot system. This research was conducted to improve the Kilobot hardware design for reliable construction and experiment performance based on our building and debugging experience. The issues we encountered included unexpected short and open circuits when the boards were out of the reflow oven, potentially limited memory at run time, as well as the strict lighting condition (in a dark room) to conduct phototaxis movement.

For debugging, we provided some guidelines and tips that we figured out while building Kilobots. The previous WCU Kilobot version 1.1 created in 2016 was done in PADS, for which we no longer had the license, and hence we redid the schematics and PCB layout in Altium Designer in version 1.2, with added spacing between crowded components to ease the building process. Version 1.2 was still in the design phase without prototyping yet but it made the later revisions in Altium Designer possible.

For ambient light sensitivity, we experimentally determined the revised emitter resistance and verified it in the move-away-from-light movement, in a relatively brighter environment in the room light condition than in a dark room. This version 1.3 was to swap out one component from version 1.1 and we could conduct physical experiments using them.

For the potential limited memory issue, we conducted a comparison of the properties of the microcontrollers and updated the PCB of the Kilobot with a new microcontroller ATmega1284, which contains 128 KB of flash memory. The new microcontroller also allowed the addition of a second ambient light sensor. With it, the Kilobot could tell its orientation relative to the light source improving its controllability. This subsequent version 1.4 was also

still in the design phase without prototyping, but it addressed two main issues from all the earlier experiments, and it laid a solid foundation for future design work.

In the future, we will explore the required steps to update the library for programming and prototype the WCU Kilobot version 1.4. We still have components purchased earlier that could be used to build version 1.4. We will then continue carrying out shape formations in both additive and subtractive approaches and possibly fuse the two approaches to achieve both accuracy and speed.

REFERENCES

- [1] E. Şahin, “Swarm robotics: From sources of inspiration to domains of application,” in *Lecture Notes in Computer Science*, 2005, vol. 3342, pp. 10–20.
- [2] F. Mondada, E. Franzi, and A. Guignard, “The Development of Khepera,” *Proc. 1st Int. Khepera Work.*, vol. 64, pp. 7–14, 2007.
- [3] C. Cianci *et al.*, “The e-puck, a Robot Designed for Education in Engineering,” *Proc. 9th Conf. Auton. Robot Syst. Compet.*, vol. 1, no. 1, pp. 59–65, 2009.
- [4] S. Kornienko and S. Kornienko, “IR-based Communication and Perception in Microrobotic Swarms,” *7th Work. Collect. Swarm Robot.*, pp. 1–15, 2010.
- [5] P. Valdastrì *et al.*, “Micromanipulation, communication and swarm intelligence issues in a swarm microrobotic platform,” *Rob. Auton. Syst.*, vol. 54, no. 10, pp. 789–804, 2006.
- [6] F. Mondada, L. M. Gambardella, D. Floreano, S. Nolfi, J. L. Deneubourg, and M. Dorigo, “The cooperation of swarm-bots: Physical interactions in collective robotics,” *IEEE Robot. Autom. Mag.*, vol. 12, no. 2, pp. 21–28, Jun. 2005.
- [7] A. Emre, T. Ku Leuven, H. Çelikkanat, and L. Bayindir, “Kobot: A mobile robot designed specifically for swarm robotics research Swarm robotics View project Collective motion in robotics View project,” 2007.
- [8] J. D. McLurkin, “Stupid robot tricks: A behavior-based distributed algorithm library for programming swarms of robots.” Massachusetts Institute of Technology, 2004.
- [9] M. Rubenstein, C. Ahler, and R. Nagpal, “Kilobot: A low cost scalable robot system for collective behaviors,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2012, pp. 3293–3298.
- [10] M. Rubenstein, A. Cornejo, and R. Nagpal, “Programmable self-assembly in a thousand-

- robot swarm,” *Science* (80-.), vol. 345, no. 6198, 2014.
- [11] “Kilobot @ WCU |.” [Online]. Available: <https://kilobot.wcu.edu/>. [Accessed: 19-Apr-2020].
- [12] “Kilobotics.” [Online]. Available: <https://www.kilobotics.com/>. [Accessed: 20-Apr-2020].
- [13] “Kilobotics (Kilobot Image).” [Online]. Available: <https://www.kilobotics.com/documentation>. [Accessed: 20-Apr-2020].
- [14] “kilobot - Recent models | 3D CAD Model Collection | GrabCAD Community Library.” [Online]. Available: <https://grabcad.com/library?utf8=√&query=kilobot>. [Accessed: 20-Apr-2020].
- [15] “Hardware - Overhead Controller - Kilobots.” [Online]. Available: https://diode.group.shef.ac.uk/kilobots/index.php/Hardware_-_Overhead_Controller. [Accessed: 24-Apr-2020].
- [16] “WCU OHC Image.” [Online]. Available: <https://kilobot.wcu.edu/wp-content/uploads/2017/03/Calibration.jpg>. [Accessed: 20-Apr-2020].
- [17] M. Gauci, R. Nagpal, and M. Rubenstein, “Programmable Self-disassembly for Shape Formation in Large-Scale Robot Collectives,” in *Springer*, 2018, pp. 573–586.
- [18] A. Tahabilder and Y. Yan, “Building Kilobots In-house for Shape-Formation,” *IEEE SoutheastCon*, 2020.
- [19] N. Thomas, Y. Yan, and H. Jack, “Maker: A kilobot swarm,” in *123rd ASEE (American Society of Engineering Education) Annual Conference and Exposition*, 2016.
- [20] N. Thomas and Y. Yan, “Make Kilobots truly accessible to all the people around the world,” in *Proceedings of IEEE Workshop on Advanced Robotics and its Social Impacts, ARSO*, 2016, vol. 2016-Novem, pp. 43–48.

[21] “Lux Light Meter Free - Apps on Google Play.” [Online]. Available:
https://play.google.com/store/apps/details?id=com.doggoapps.luxlight&hl=en_US.
[Accessed: 22-Apr-2020].

APPENDIX

Kilobot ambient light sensor test code using Arduino Uno

```
//Author: Anik Tahabilder

//Name: Kilobot debugging using arduino uno for ambient light sensitivity.

//Date: 12/02/2019

void setup() {

  // enable serial output

  Serial.begin(9600);

}

void loop() {

  // read ADC and convert to Voltage

  int adcValue1 = analogRead(A0);

  int adcValue2 = analogRead(A1);

  int adcValue3 = analogRead(A2);

  int adcValue4 = analogRead(A3);

  int adcValue5 = analogRead(A4);

  int adcValue6 = analogRead(A5);

  Serial.println( "\t" "\t" "Measured1=" + String(adcValue1) + "\t" "\t" "Measured2=" + String(adcValue2)

+ "\t" "\t" "Measured3=" + String(adcValue3) + "\t" "\t" "Measured4=" + String(adcValue4) + "\t" "\t"

"Measured5=" + String(adcValue5) + "\t" "\t" "Measured6=" + String(adcValue6));

  // wait

  delay(100);

}
```

Kilobot code for ambient Light sensitivity

```
#include <kilolib.h>
#define DEBUG
#include <debug.h>
int current_light = 0;
void sample_light()
{
    // The ambient light sensor gives noisy readings. To mitigate this,
    // we take the average of 300 samples in quick succession.
    int number_of_samples = 0;
    uint32_t sum = 0;
    while (number_of_samples < 100)
    {
        int sample = get_ambientlight();
        // -1 indicates a failed sample, which should be discarded.
        if (sample != -1)
        {
            sum = sum + sample;
            number_of_samples = number_of_samples + 1;
        }
    }
    // Compute the average.
    current_light = sum / number_of_samples;
    delay(500);
}
void setup() { }
// print voltage every second
void loop() {
    sample_light();
    delay(100);
    set_color(RGB(1, 1, 0));
}
```

```
    delay(100);
    set_color(0, 0, 0);
    printf("Light value\n");
    printf("%d\n", current_light);
    //printf("%f\n", current_light);
    //printf("%u\n", current_light);
}
int main() {
    kilo_init();
    debug_init();
    kilo_start(setup, loop);
    return 0;
}
```

Kilobot Code move away from light

```
#include <kilolib.h>
// Constants for light following.
#define THRESH_LO 200
#define THRESH_HI 900
// Constants for motion handling function.
#define STOP 0
#define FORWARD 1
#define LEFT 2
#define RIGHT 3
int current_motion = STOP;
int current_light = 0;
int previous_light=0;
// Function to handle motion.
void set_motion(int new_motion)
{
    // Only take an action if the motion is being changed.
    if (current_motion != new_motion)
    {
        current_motion = new_motion;

        if (current_motion == STOP)
        {
            set_motors(0, 0);
        }
        else if (current_motion == FORWARD)
        {
            spinup_motors();
            set_motors(kilo_straight_left, kilo_straight_right);
        }
        else if (current_motion == LEFT)
```

```

    {
        spinup_motors();
        set_motors(kilo_turn_left, 0);
    }
    else if (current_motion == RIGHT)
    {
        spinup_motors();
        set_motors(0, kilo_turn_right);
    }
}
}
// Function to sample light.
void sample_light()
{
    // The ambient light sensor gives noisy readings. To mitigate this,
    // we take the average of 300 samples in quick succession.
    int number_of_samples = 0;
    uint32_t sum = 0;
    while (number_of_samples < 300)
    {
        int sample = get_ambientlight();
        // -1 indicates a failed sample, which should be discarded.
        if (sample != -1)
        {
            sum = sum + sample;
            number_of_samples = number_of_samples + 1;
        }
    }
    // Compute the average.
    current_light = sum / number_of_samples;
}

```

```

void setup()
{
  // This ensures that the robot starts moving.
  set_motion(LEFT);
}
void loop()
{
  sample_light();
  //changes behaviour of Kilobots based on light intensity
  if (current_light >= THRESH_HI){
    set_color(RGB(1, 1, 1));
    set_motion(FORWARD);
  }
  else if (current_light > previous_light)
  {
    // Generate an 8-bit random number (between 0 and 2^8 - 1 = 255).
    int random_number = rand_hard();

    // Compute the remainder of random_number when divided by 4.
    // This gives a new random number in the set {0, 1, 2, 3}.
    int random_direction = (random_number % 4);

    // There is a 50% chance of random_direction being 0 OR 1, in which
    // case set the LED green and move forward.
    if ((random_direction == 0) || (random_direction == 1))
    {
      set_color(RGB(0, 1, 1));
      set_motion(FORWARD);
      delay(1000);
    }
    // There is a 25% chance of random_direction being 2, in which case

```



```

// set the LED red and move left.
else if (random_direction == 2)
{
    set_color(RGB(1, 0, 0));
    set_motion(LEFT);
    delay(1000);
}
// There is a 25% chance of random_direction being 3, in which case
// set the LED blue and move right.
else if (random_direction == 3)
{
    set_color(RGB(0, 0, 1));
    set_motion(RIGHT);
    delay(1000);
}
}
else if (current_light < previous_light)
{
    set_color(RGB(0, 1, 0));
    set_motion(FORWARD);
    delay(1000);
}
previous_light=current_light;
}
int main()
{
    kilo_init();
    kilo_start(setup, loop);
    return 0;
}

```