Systems and Control Engineering Laboratory (SC 626)
Kilobotics

Abhishek Rajopaadhye (193230004)
Guntaka Harsha Priyanka (193236001)
Neelam Patwardhan (193234001)

February 18, 2020

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

Kilobots(Figure 1.1) are low cost robots designed at Harvard University's Self-Organizing Systems Research Lab http://www.eecs.harvard.edu/ssr. The robots are designed to make testing collective algorithms on hundreds or thousands of robots accessible to robotics researchers.

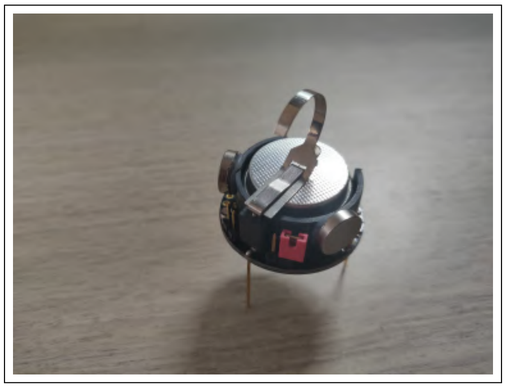

Figure 1.1: Kilobot

Though the Kilobots are low-cost, they maintain abilities similar to other collective robots. These abilities include differential drive locomotion, on-board computation power, neighbor-to-neighbor communication, neighbor-to neighbor distance sensing, and ambient light sensing. Additionally they are designed to operate such that no robot requires any individual attention by a human operator. This makes controlling a group of Kilobots easy, whether there are 10 or 1000 in the group.

1.2 Features of Kilobot

- Processor : ATmega 328p (8bit @ 8MHz)

- Memory :
  - 32 KB Flash used for both user program and bootloader
  - 1KB EEPROM for storing calibration values and other non-volatile data and 2KB SRAM.
- Battery : Rechargeable Li-Ion 3.7V, for a 3 months autonomy in sleep mode. Each Kilobot has a built-in charger circuit, which charges the onboard battery when +6 volts is applied to any of the legs, and GND is applied to the charging tab.
- Charging : Kilobot charger for 10 robots simultaneously (optional).
- Communication : Kilobots can communicate with neighbors up to 7 cm away by reflecting infrared (IR) light off the ground surface.(Figure 1.2)
- Sensing : 1 IR and 1 light intensity.
  - When receiving a message, distance to the transmitting Kilobot can be determined using received signal strength. The distance depends on the surface used as the light intensity is used to compute the value.
  - The brightness of the ambient light shining on a Kilobot can be detected.
  - A Kilobot can sense its own battery voltage.
- Movement : Each Kilobot has 2 vibration motors, which are independently controllable, allowing for differential drive of the robot. Each motor can be set to 255 different power levels.
- Light : Each Kilobot has a red/green/blue (RGB) LED pointed upward, and each color has 3 levels of brightness control.
- Software : The Kilobot Controller software (kiloGUI) is available for controlling the controller board, sending program files to the robots and controlling them.
- Programming : For programming, the open source development software WinAVR combined with Eclipse gives a C programming environment. An API with basic functions such as motor speed, led control, distance measurement is available and some examples are provided.
- Dimensions : diameter: 33 mm, height 34 mm (including the legs, without recharge antenna).

1.3 Requirements

- Hardware :
  - Computer with Microsoft Windows and an USB port (not included)
  - Kilobot robot
  - Over-head controller (OHC)
  - Kilobot charger

- Software : To start programming the Kilobot with the new version from kilobotics, we have two solutions.
  - Online editor https://www.kilobotics.com/editor
  - Install WinAVR and Eclipse to compile the whole library on your computer https://github.com/mgauci/kilobot_notes/blob/master/eclipse_winavr_setup/eclipse_winavr_setup.md

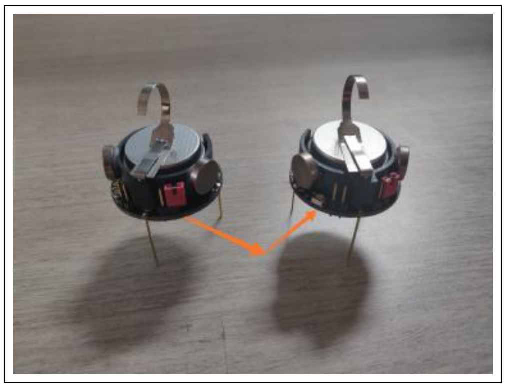

Figure 1.2: IR sensing

1.4 Over-Head Controller(OHC)

In case of Kilobots, instead of plugging in a charging cable for each robot in order to update its program, each can receive a program via an IR communication channel. This allows an over head IR transmitter to program all the robots in the collective in a fixed amount of time, independent of the number of robots.

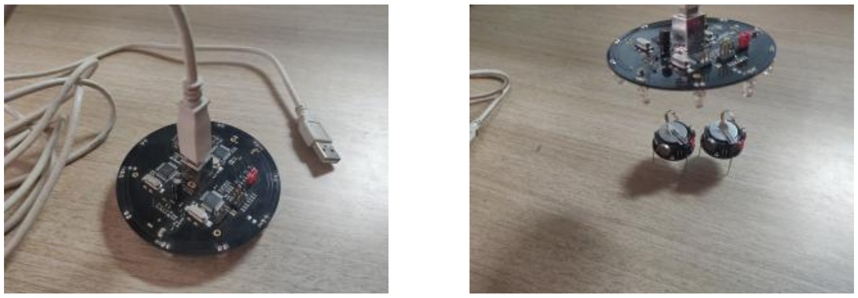

Figure 1.3: Overhead Controller

1.5 Motor Calibration

Kilobot should be operated on a smooth, flat surface to ensure proper robot mobility. Only one Kilobot can be calibrated at the same time.

1. Place the Kilobot in PAUSE mode.
2. Open KiloGUI interface and then open Calibration mode.(Figure 1.4)
3. The first line (Unique ID) can be used to save an ID in your Kilobot. This can be useful if you want to save all calibration value for each Kilobot.
4. The second line (Turn Left) will configure the kilo_turn_left parameter to set the CCW movement. Set a value for the motor (approximately 70) and press the Test button. Adjust the value to obtain a smooth move.
5. Do the same as explain in point 4, for the right motor (Turn Right line).
6. Next is to set the straight move parameters. Start with the same value for the left and right motor (approximately 60) and press the Test button. Now adjust the two value to move the Kilobot as straight as possible.
7. Finally, when all the parameters are fine, you can press the Save button to write the parameters in the EEPROM of the Robot.

Chapter 2

Orbiting of Kilobots

2.1 Objective

Our objective is to make a Kilobot(planet) orbit around

- one stationary Kilobot(star)
- multiple stationary Kilobots.

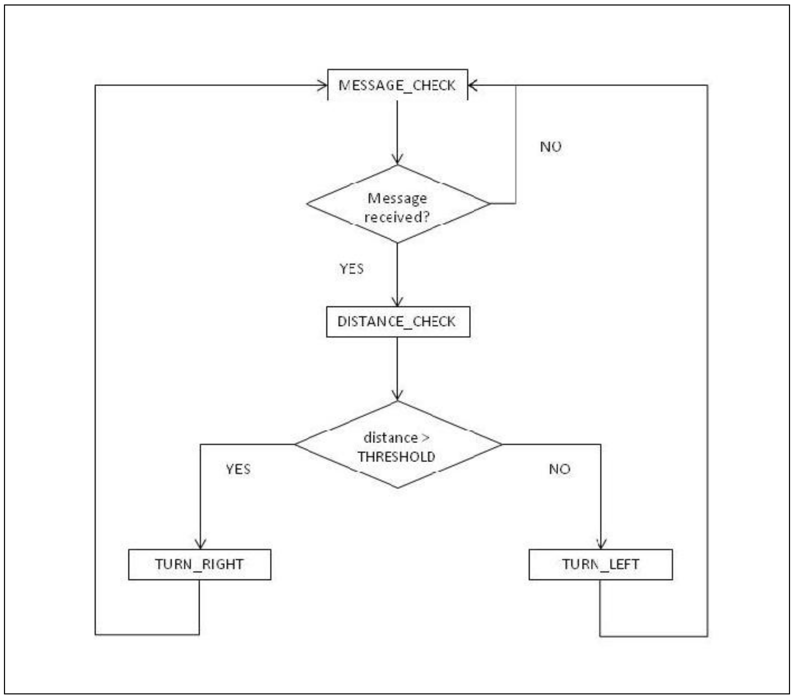

Figure 2.1: Flowchart for orbiting a Kilobot(Single star)

2.2 With single stationary Kilobot

The algorithm for the planet motion with a single star is as follows:

1. Check for the message signal from star Kilobot.
2. If message not received go to step 1.
3. If message is received, calculate the distance.
4. If the distance is greater than Threshold value(fixed radius), move right. Else, move left.
5. Go to step 1.

Flowchart to the corresponding algorithm is illustrated in Figure 2.1.

Results and Demonstration

As per the Kilobotics manual, the maximum and minimum communication ranges are 110mm and 33mm respectively. So we have taken the orbit radius (*THRESHOLD*) as 50mm, which falls within good communication range. Also, we have given motor nn time (*MOTOR_ON_DURATION*) as 500ms.

Video of working demo of problem statement can be accessed from the link in Figure 2.2.

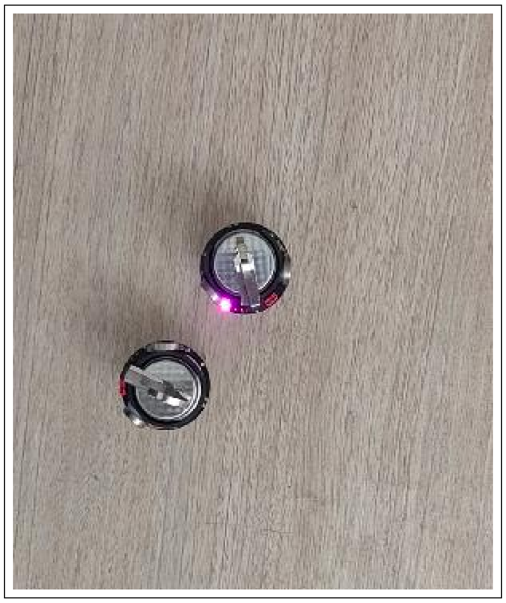

Figure 2.2: Orbiting of Kilobot (Single Star)

2.3 With multiple stationary Kilobots

We have placed one more Kilobot within the communication range of the planet and star. It was observed that the planet collides with one of the star due to communication delay between them.

Video of working demo can be accessed from the link in Figure 2.3.

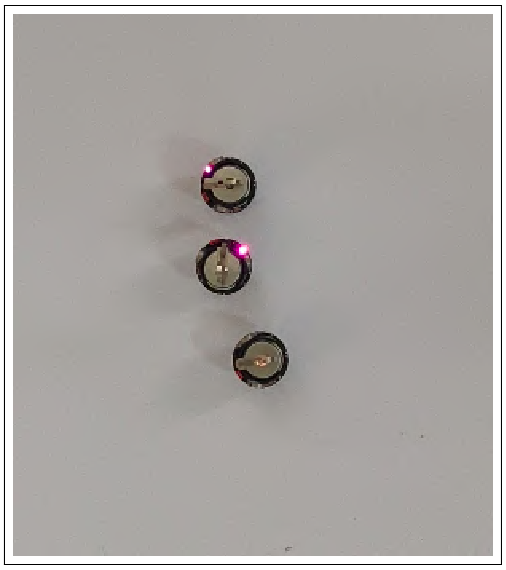

Figure 2.3: Planet colliding with one of the stars

To avoid collision, we have modified the above algorithm by checking the message from the neighbor Kilobots for multiple times. The algorithm is as follows:

1. First initialize a variable (*distance*) corresponding to distance between star and planet to 1000 (or to any other value which is greater than the range of Kilobots).
2. Check for message from neighbor Kilobots. If you receive the message, estimate the distance, then compare it with the value in the *distance* variable and replace the variable with the lowest value.
3. Go to step 2, until the number of messages received are equal to *TOTAL_NUM_COMMUNICATION*.
4. Now, the value of *distance* variable will be our distance between star and planet. If the distance is greater than required orbit radius (*THRESHOLD*) move right. Else, move left.
5. Go to step 1.

The flowchart for the corresponding algorithm is illustrated in Figure 2.4.

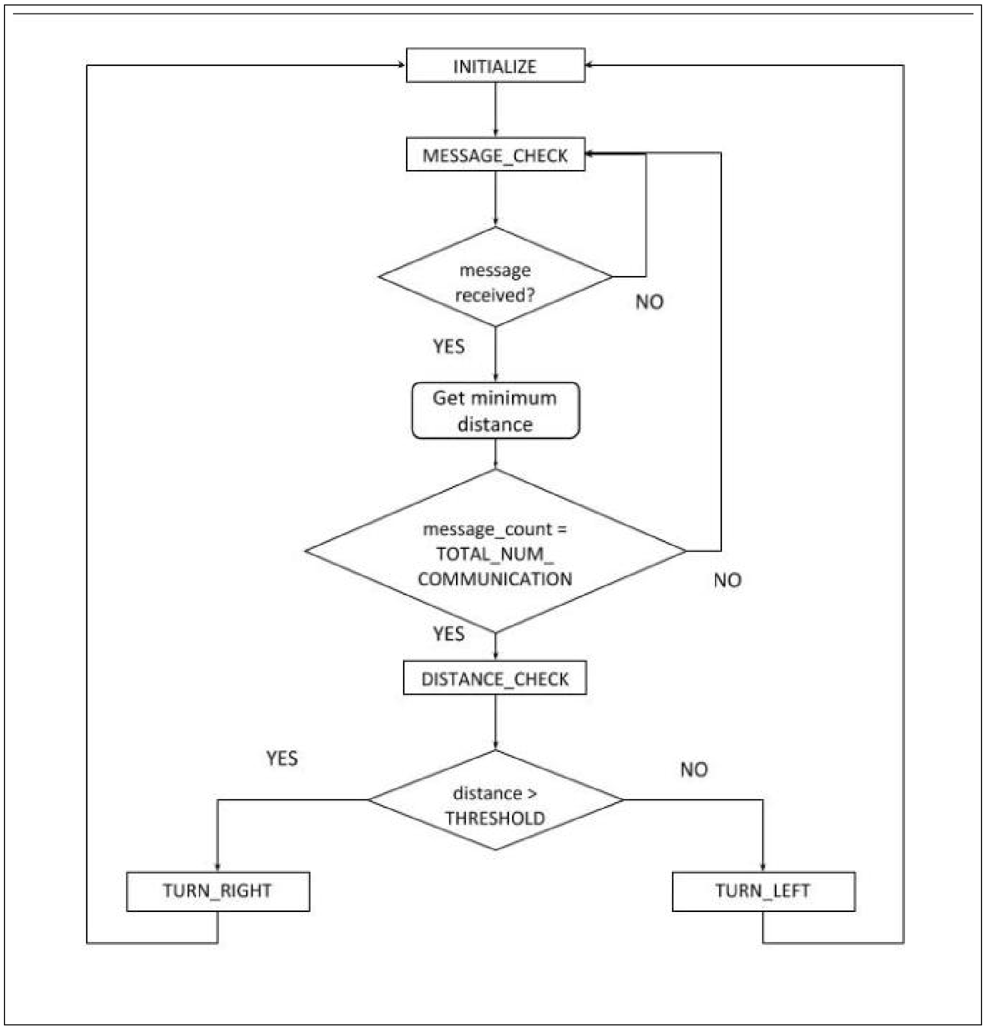

Figure 2.4: Flowchart for orbiting a Kilobot(Multiple star)

Results and Demonstration

We have used the same orbit radius (*THRESHOLD = 50*), and same motor on time (*MOTOR_ON_DURATION = 500*) as in the case of single star. To avoid collision between Kilobots we have checked for the message for multiple times i.e. *TOTAL_NUM_COMMUNICATION = 4*.

Video of working demo for this problem statement can be accessed from the link in Figure 2.5.

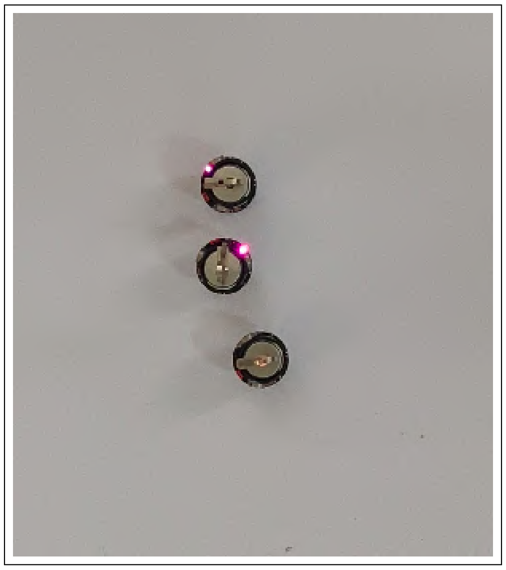

Figure 2.5: Orbiting of Kilobot (Multiple Star, *MOTOR_ON_DURATION = 500*, *TOTAL_NUM_COMMUNICATION = 4*)

It can be observed that in this case the speed of the planet Kilobot is less compared to the case with single stationary Kilobot. It can be increased by increasing the *MOTOR_ON_DURATION* and decreasing the *TOTAL_NUM_COMMUNICATION*.

Video of working demo for this problem statement can be accessed from the link in Figure 2.6.

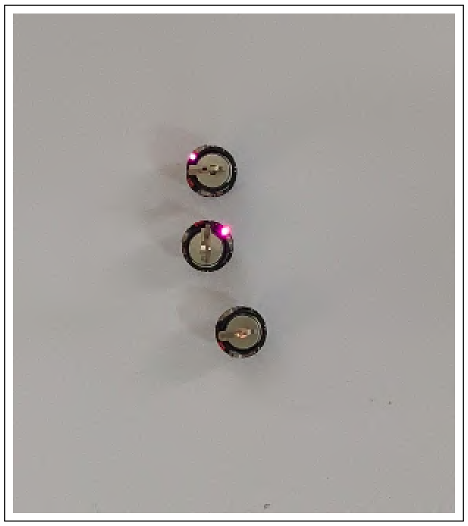

Figure 2.6: Orbiting of Kilobot (Multiple Star, *MOTOR_ON_DURATION = 800*, *TOTAL_NUM_COMMUNICATION = 3*)

Chapter 3

Gradient Formation

3.1 Objective

Our objective of these kilobots is to form a desired shape as per the assigned id values;the farthest one being the first to move and so on. The algorithm is as follows:

1. First, we initialize the reference kilobot to 0. This kilobot is going to send the message.
2. The id's of the other kilobots are assigned based on the distance threshold.
3. The bot(s) nearest to the reference bot will receive the message and get id 1.
4. The self unique id of all the other kilobots is preset at 251.
5. These bots send data and those within the zone of communication with threshold set as 50mm;their unique ids will be checked and updated to +1.If not, the message is checked again.
6. If there are multiple kilobots within the same zone of communication and falling within the same threshold limit,the one(s) having the least distance are again checked for their id's.
7. The temporary id is compared to the present self unique id and the one with the least id is updated as the new id of the kilobot(s).
8. To distinguish between set of kilobots having same id's,they are represented with colors red,blue,green etc.

Flowchart to the corresponding algorithm is illustrated in Figure 3.1.

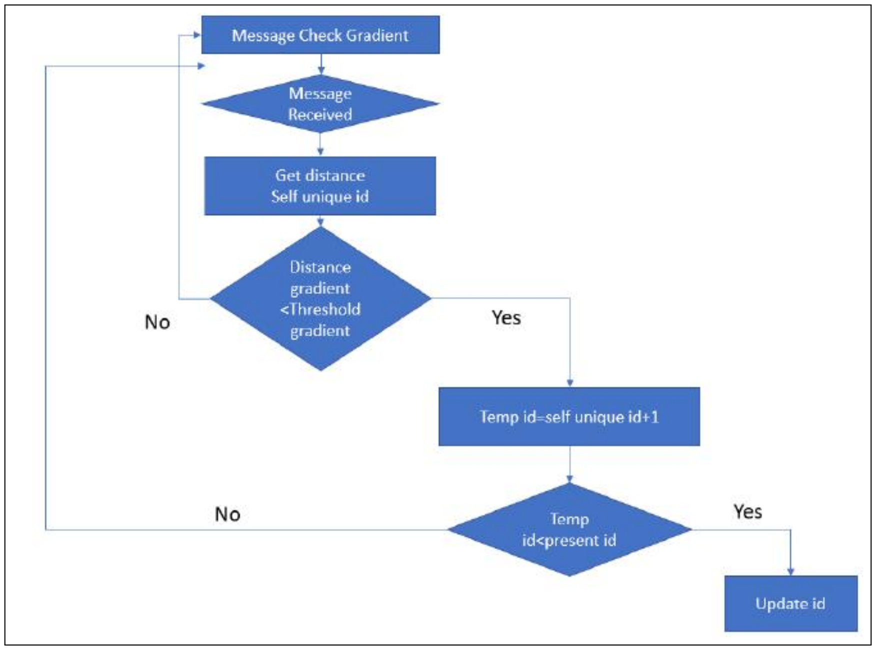

Figure 3.1: Flowchart for id assignment

3.2 Results and Demonstration

To differentiate between different ids of the kilobots they are converted into binary to represent three colors red, blue and green. Video of working demo of problem statement can be accessed from the link in Figure 3.2.

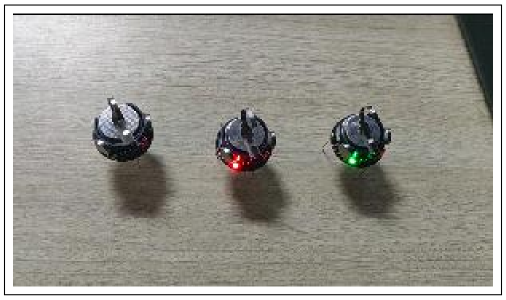

Figure 3.2: Display of colors as per different ids

Chapter 4

Edge following

4.1 Introduction

Our objective is to make the kilobots which are on the outer edge to move along the edge of a group of kilobots by measuring distances without being physically blocked and reach the reference bot.

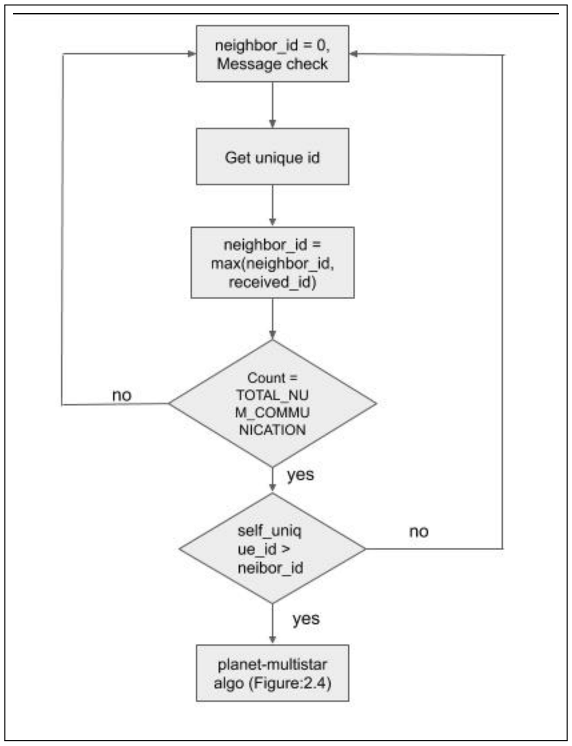
neighbor_id = 0, Message check
Get unique id
neighbor_id = max(neighbor_id, received_id)
Count = TOTAL_NUM_COMMUNICATION
no
yes
self_unique_id > neibor_id
no
yes
planet-multistar algo (Figure:2.4)

Figure 4.1: Flowchart for Edge following

The algorithm is as follows:

1. Check for the message from neighboring kilobots for TOTAL_NUM_COMMMUNICATION times.
2. For each message received get the unique id and store the maximum id.
3. Compare it's own unique id with maximum neighbor id.
4. If self unique id ¿ max. neighbor id, start moving towards the reference bot using *planet multistar* algorithm.
5. If not, go to step 1.

Flowchart to the corresponding algorithm is illustrated in Figure 4.1.

4.2 Result and Demonstration

Kilobots along the outer edge of the initial group are able to move without being physically blocked. They can determine that they are on the outer edge by comparing their gradient values to those of their neighbors. These robots can then use edge-following around the stationary initial group to reach the reference kilobot. Video of working demo of problem statement can be accessed from the link in Figure 4.2.

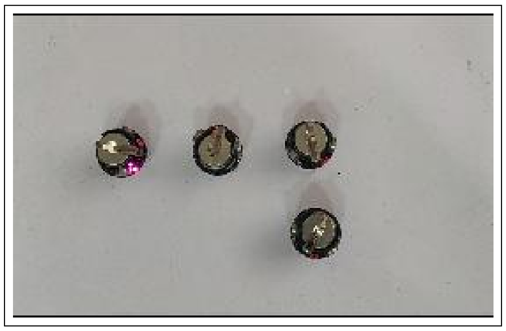

Figure 4.2: Edge Following

It can be observed that once the outer edge (with max. unique id) kilobot goes out of communication range of the next outer edge kilobot, it will start moving towards the reference bot.