<Undergraduate Thesis: EEE-2018-2-00-00>

Various Swarm Behavior Inspired by Nature Using Vibration Locomotion Robots

Jiseung Jung, Jaewoo Jung
School of Electrical and Electronic Engineering
College of Engineering
Yonsei University

<Undergraduate Thesis: EEE-2018-2-00-00>

Various Swarm Behavior Inspired by Nature Using Vibration Locomotion Robots

Thesis Advisor: DaeEun Kim

A thesis submitted in a partial fulfillment
for the senior Creative Independent Study's requirements

December 2018

Jiseung Jung, Jaewoo Jung
School of Electrical and Electronic Engineering
College of Engineering
Yonsei University

Acknowledgment

연구실에서 인턴으로 들어와서 1년동안 그리고 대학생으로 시간이 지나 졸업을 맞게 되었습니다. 그 동안의 연구의 결실로 나온 졸업논문이 완성되기까지 도움을 주신 많은 분들에게 감사의 글을 전달하려고 합니다.
가장 먼저 저희에게 꾸준한 피드백과 도움을 주신 김대은 지도교수님께 감사드립니다. 또한 저희 연구실 인원들께 감사를 드립니다. 졸업연구 조교를 맡아준 김민철 학생과 발표 준비를 도와준 강병문 학생에게 감사합니다. 긍정적인 관심과 응원을 해준 최종하, 손충현, 이창민, 백의현, 김슬기, 김만동, 김진수 학생에게 감사를 드립니다.
또한 항상 저에게 끊임없는 조언과 격려를 해주시고 재정적으로도 지원을 아끼지 않으신 어머니, 아버지께 감사를 드립니다.

Contents

Figure index

Table index

ABSTRACT

Various Swarm Behavior Inspired by Nature Using Vibration Locomotion Robots

In nature, animals have evolved and adapted to survive in their environment by means of various ways. We are particularly interested in the unique swarm behavior displayed by animals such as ants, fish schools and flocks of birds. Each entity follows a specific set of rules which when combined across the group emerges as a coordinated activity called ‘swarming’. By mimicking this behavior, we can find new applications and solutions to complex problems. We can also form a better understanding of nature and the animals it inhabits.

In this paper, we attempt to incorporate the behavior of animal swarms into robots. We start by designing vibration robots then propose a specified method of control. For hardware, we define 3 specifications that must be met to design swarm robots. Through multiple experimentation along with trial and error we were able to make a robot that met these specifications. For software, we were able to localize the robot using computer vision, define a method of locomotion that best suited vibration robots and propose a modified version of a well-known algorithm, DWA*, that is specific to our application.

We verify our system by mimicking a well-known form of swarm behavior which is formation and further show potential for various other biomimetic swarm behavior.

Key words : swarm behavior, vibration locomotion, swarm robot, biomimicry, collision avoidance, arrival to desired location, formation, computer vision, control algorithm

1. Introduction

As humans we are very familiar with terms such as ‘group’, ‘community’, or ‘cooperation’. We live in a society where it is nearly impossible to survive alone. We call ourselves ‘social animals’ and tend to think we know much about living together. After all we have evolved several centuries learning to do so. However, we are not the only ones. There are also many others who have evolved alongside us adapting their own method of living together. These are our animal counterparts and we are far from fully understanding them. The collective behavior exhibited by animals is called ‘swarming’. Animal swarms behave in distinct and unique ways that very much differ from familiar human behavior. By mimicking this behavior, we can find new applications and solutions to various problems. We can also form a better understanding of nature and the animals it inhabits.

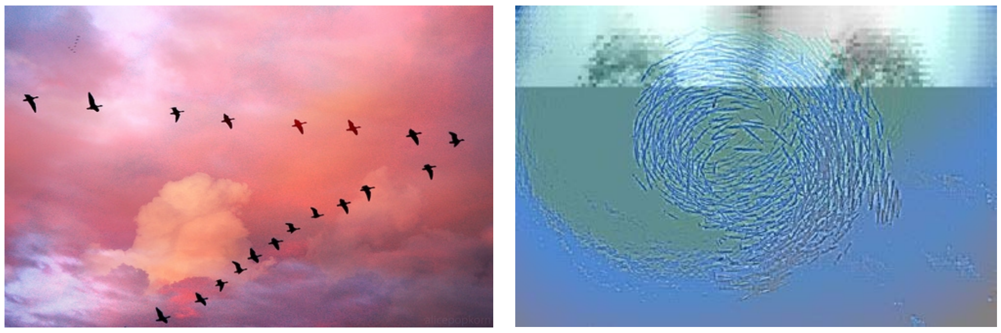
Robots and artificial intelligence have been a long endearing dream for humans. Whether it be to lessen the labor that is burdened upon us or to satisfy our longing to be the creators of artificial life there have been various attempts to accomplish this goal. Amongst these attempts we are very interested in the field of biomimetics (the imitation of the models, systems, and elements of nature). In this paper, we attempt to incorporate the behavior of animal swarms into robots. We start by designing vibration robots then propose a specified method of control.

2. Background

In this section we elaborate on the background of our research. Specifically, the swarm behavior that is found in nature and its characteristics, previous attempts to mimic this behavior and the development framework that we used to implement our design.

2.1. Swarm behavior in nature

In nature, animals have evolved and adapted to survive in their environment by means of various ways. Some of these have baffled researchers by their uniqueness and creativity. We are particularly interested in swarm behavior displayed by animals such as ants, fish schools and flocks of birds. Each entity follows a specific set of rules which when combined across the group emerges as a coordinated activity. This coordination although performed locally has the efficiency similar to that of a globally planned scheme. Ants for example follow the simple rule of sharing their food with neighbors. Consistence with multiple entities eventually allows the entire group to be fed. Also, flocks of birds can maintain a formation by positioning themselves locally based on their neighbors. As such even the most complex tasks can be broken down into a continuity of simple subtasks. We wish to exploit this aspect of swarm behavior.



(a) (b)

Fig. 2.1. Collective behavior of animals (a) V-formation of migratory birds (b) Schools of fish

2.2. Implementation of swarm robots

Various attempts have been made in the research of swarm behavior. A well-known application is that of the warehouse swarm robots. Multiple robots move in a unified fashion to accomplish the complicated task of organizing warehouse merchandise efficiently. Another application is formation. Various robots such as drones are coordinated to form specific shapes. We were mostly inspired by Kilobot, a robot which is controlled with vibration motors. We used Kilobot as a baseline to develop an improved version of vibration locomotion robots.

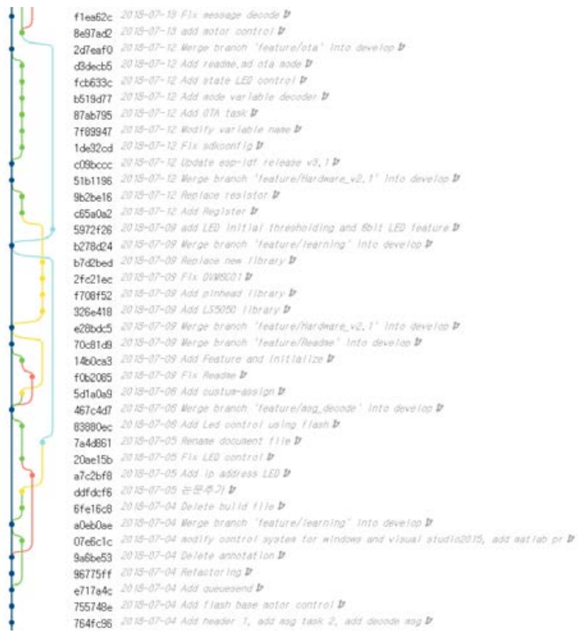
2.3. Development Frameworks

While we were working on our thesis, we used these develop frameworks. We used an Eagle CAD from Autodesk for drawing schematics and PCB.

For developing the firmware, we used ESP-IDF (Espressif IoT Develop Framework). It is developed in Linux OS environment, and most of the code is made up of C / C++, except for the application codes ex) firmware upload, and serial monitor are made up of python. We were developed in Window OS environment, so we used Cygwin which is a virtual Linux environment.

For software, we used MATLAB because it supports TCP/IP communication and it has many toolboxes for image processing.

Lastly, we used Git and Github for version control. Git has many advantages for co-work. Using Git-flow branch model, we could develop more efficiently. Figure 2.2 is the Git flow of our repository



```
f1ea62c 2018-07-13 Fix message decode
8e97ad2 2018-07-13 add motor control
2d7eaf0 2018-07-12 Merge branch 'feature/ota' into develop
d9decb5 2018-07-12 Add readme.md ota mode
fcb633c 2018-07-12 Add state LED control
b519d77 2018-07-12 Add mode variable decoder
87ab795 2018-07-12 Add OTA task
7f89947 2018-07-12 Modify variable name
1de32cd 2018-07-12 Fix sdkconfig
c09bccc 2018-07-12 Update esp-idf release v3.1
51b1196 2018-07-12 Merge branch 'feature/Hardware_v2.1' into develop
9b2be16 2018-07-12 Replace resistor
c65a0a2 2018-07-12 Add Register
5972f26 2018-07-09 add LED initial thresholding and 8bit LED feature
b278d24 2018-07-09 Merge branch 'feature/learning' into develop
b7d2bed 2018-07-09 Replace new library
2fc21ec 2018-07-09 Fix OV7670
f708f52 2018-07-09 Add pinhead library
326e418 2018-07-09 Add LS5050 library
e28bdc5 2018-07-09 Merge branch 'feature/Hardware_v2.1' into develop
70c81d9 2018-07-09 Merge branch 'feature/Readme' into develop
14b0ca3 2018-07-09 Add Feature and initialize
f0b2065 2018-07-09 Fix Readme
5d1a0a9 2018-07-06 Add custum-assign
467c4d7 2018-07-06 Merge branch 'feature/msg_decode' into develop
83880ec 2018-07-06 Add Led control using flash
7a4d861 2018-07-05 Rename document file
20ae15b 2018-07-05 Fix LED control
a7c2bf8 2018-07-05 Add ip address LED
ddfdcf6 2018-07-05 논문추가
6fe16c8 2018-07-04 Delete build file
a0eb0ae 2018-07-04 Merge branch 'feature/learning' into develop
07e6c1c 2018-07-04 modify control system for windows and visual studio2015, add matlab pr
9a6be53 2018-07-04 Delete annotation
96775ff 2018-07-04 Refactoring
e717a4c 2018-07-04 Add queuesend
755748e 2018-07-04 Add flash base motor control
764fc96 2018-07-04 Add header 1, add msg task 2, add decode msg
```

Fig. 2.2. Git flow of repository

3. Hardware

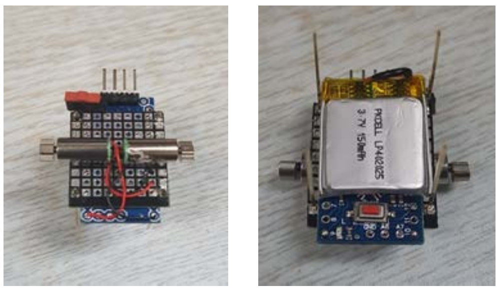
In order to design the hardware of our swarm robot, various aspects must be considered. First, the robot should be able to move freely. Second, communication between the robot and the server or communication between the robots themselves should be possible. Finally, the swarm robots should be capable of mass production. Based on this, the design was verified and fabricated through various experiments.

3.1. Hardware progression

The hardware specification was determined through the above description, and we started developing the hardware. The hardware was built 5 times in total, and the previous generation hardware was modified and supplemented.

3.1.1. Testing of vibration motor

Before making the robot, we needed to test how much maneuverability and control was possible of the vibration motors. So, we made a simple test robot shown figure 3.1 using two vibration motors, a Li-ion battery and an Arduino Pro mini. Using the robot, we did experiments for vibration control. Then we confirmed that the robot could be controlled through vibration motors.



(a) (b)

Fig. 3.1. Test Vibration robot (a) Top view (b) Bottom view

3.1.2. 1st Hardware

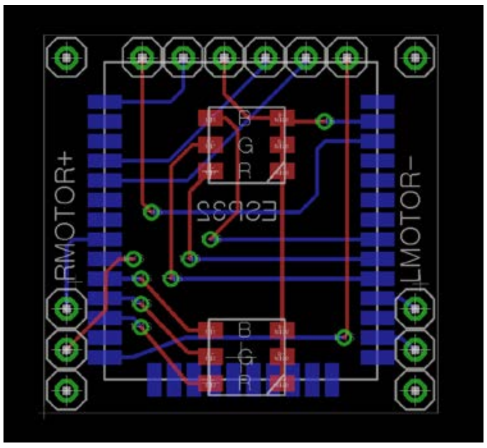
After checking the maneuverability of the vibration motors, we selected the microcontroller unit (MCU) for controlling the robot and communication. We compared three products: Arduino Pro mini, nRF52832, ESP32. Table 3.1 is a comparison table between them.

	Arduino Pro mini + ESP8266	**nRF52832**	**ESP32**
Manufactory	Atmel	Nordic Semiconductor	Espressif
Architecture	AVR	ARM	Xtensa
Operating voltage	5V	3.3V	3.3V
Clock rate	8MHz	64MHz	240MHz
Communication	Wi-Fi	RF + Bluetooth + NFC	Wi-Fi + Bluetooth

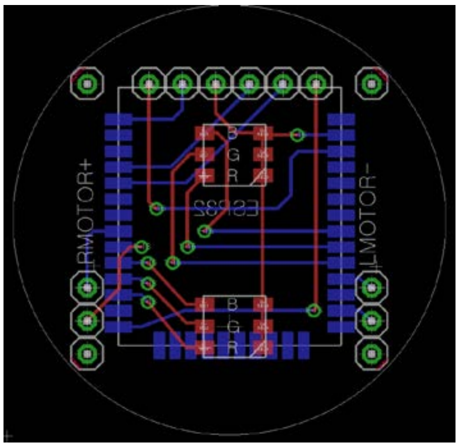
Table 3.1. Comparison table between selected MCU

Arduino Pro mini didn't support any communication, so we added ESP8266 as a communication chip. Arduino Pro mini has only 4 PWM output pins. But we needed 10 PWM output pins at least because of RGB LEDs and motors. Therefore, it wasn't suited for our system.

nRF52832 was a small module and it supported Bluetooth, RF and NFC. We tested the two methods for communication beside NFC and decided that Bluetooth did not fit the characteristics of the swarm robot since Bluetooth is limited to a maximum of 8 pairs. RF transmission speed was not fast. Of course, in the case of the nRF52832 product, a mesh network could be configured through Bluetooth 5.0 but it was not commercialized yet, and the SDK was under development which led us to decide that it was not adequate for use. Therefore, after trying and testing various modules we selected the ESP32 module which supports Wi-Fi made by Espressif. The model of the robot was designed by drawing PCB CADs with Eagle from Autodesk. We initially considered 2 types of frames (rectangular and circle) as shown in figure 3.2. We made two types of robot then we decided to use the circular design.



(a)



(b)

Fig. 3.2. 1st Hardware PCB CAD (a) Rectangular shape (b) Circle shape

We added RGB LEDs to control the robot through the camera and installed two in each position to distinguish the robot's head and tail from each other. Also, we used it to debug firmware. When using the LEDs to pinpoint the robot's position and orientation we ran into some serious LED glare issues. The glare and distortion of the LEDs on the camera lens resulted in some alterations in position or even blind spots. This was later improved by building a test bed and SW thresholding.

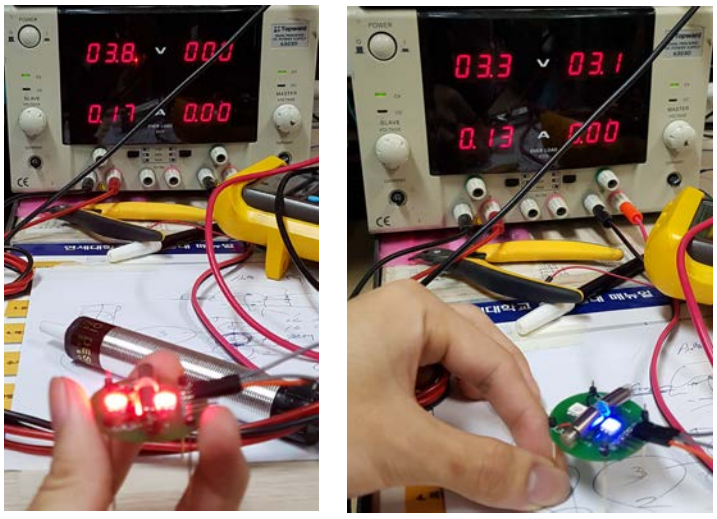
We added UART communication pins to upload firmware and debug using serial communication. It needs 6 pins VCC, GND, RX, TX, EN, RTS. VCC, GND are power source pins and RX, TX are communication pins. RTS is a reset pin so we can use it to reset firmware physically. EN is an enable pin which is always on. However, it needs to connect to DTS pin when uploading firmware.

After assembling the hardware through soldering (Figure 3.3), we measured the usage time with a power supply. By doing this we were able to roughly calculate the usage time of the battery. When using two motors, communication and LEDs, 0.17A is consumed (Figure 3.4 (a)). Since an 180mAh battery is used, it doesn't even work for one hour. So, we need to underclock the MCU clock rate from 240MHz to 160MHz considering the short use time.

(a) (b)

Fig. 3.3. 1st Hardware (a) Top-Left view (b) Bottom-Left view

After underclocking, we measured the power consumption again and discovered 0.13A was consumed (Figure 3.4 (b)). However, when the consumption of the battery is more than a certain amount, the output is decreased. Therefore, 140mAh is calculated as the capacity of the battery.



(a) (b)

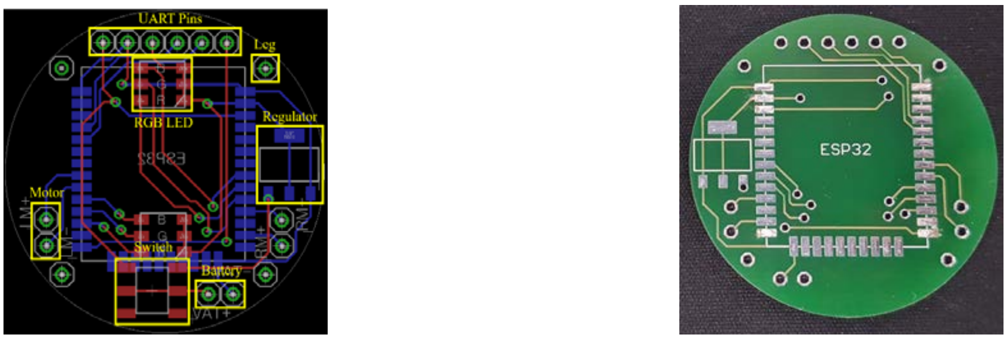
Fig. 3.4. Measuring power consumption using power supply
(a) Without underclocking (b) With underclocking

$$\text{Use Time(h)} = \frac{Battery\ Capacity(mAh)}{Battery\ Consume\ Rate(mA)} \qquad \text{(eq 3.1.)}$$

Using eq. 3.1, the result was about 1 hour. However, since we set the battery consume rate to the maximum output, the usage time can be estimated to be 1.5 hours. Also, in practice, when the operation time was measured, it was possible to operate for about 2 hours.

3.1.3. 2nd Hardware

After making the 1st Hardware, we found some additional items and added them in the 2nd Hardware. Although the maximum charging voltage of the battery was about 4.2V, the driving voltage of the MCU was 3.3V, so a 3.3V step-down regulator was added for stable power and a power switch was added (Figure 3.5 (a)). Then, we spread the parts widely along the PCB.



(a) (b) (c)

Fig. 3.5. 2nd Hardware PCB (a) PCB CAD (b) Top view (c) Bottom view

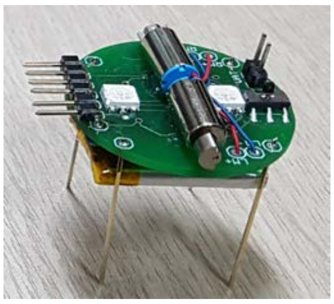


Fig. 3.6. 2nd Hardware Top-Right view

3.1.4. 3rd Hardware

After making 2nd Hardware, we found several errors. So, we fixed them and added some features in 3rd Hardware. It had 2 critical errors. First, UART pins were vulnerable and sometimes conflicted between robots while moving. When robots conflicted, it made a short-circuit between battery pins because VCC and GND in UART pins are connected to the battery directly. It didn’t cause any danger because of battery protection circuit, but the robot stopped working. Therefore, we changed the UART pins structure from 1x6 to 2x3 and moved it to the inner board.

Second, we had originally attached motors using a glue-gun manually in the 2nd Hardware. However, through experimentation we found that motor position affects movement of the robot. As a result, all the robots had different characteristics. Accordingly, we needed to fix motor position. We changed the motor with a solderable one so that we were able to fix the position of the motor using soldering.

Also, we added several features in the 3rd Hardware. We added ground planes to simplify wiring and resistors to protect the LED.

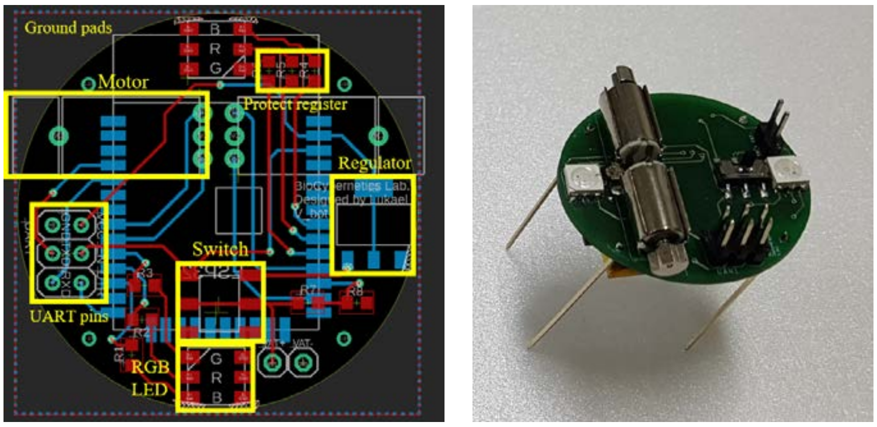
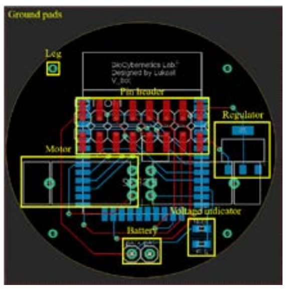


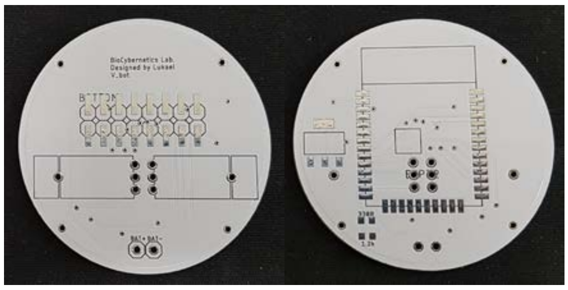
Fig. 3.7. 3rd Hardware (a) PCB CAD (b) Top-Right view

3.1.5. 4th Hardware

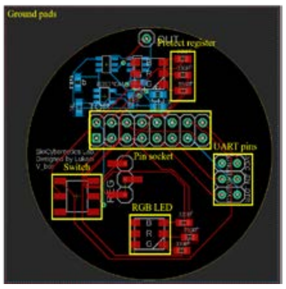
In the 4th Hardware, we once again fixed some problems and added features. First, motor conflicted between robots when moving. It caused loss of loose tracking and coordinate stealing which we discuss in detail in a later section. So, we expanded the size of board from 17.5cm to 20.5cm. Also, we used RGB LEDs and the green color board affected vision processing. Therefore, we changed the color of the board from green to white. As the board size increases, we increased battery capacity from 180mAh to 430mAh. This gave the robot more usage time. In addition, we added a battery indicator circuit that can be used to check the current capacity of the battery using a voltage divider circuit and an Analog to Digital Converter (ADC).



(a)



(b)



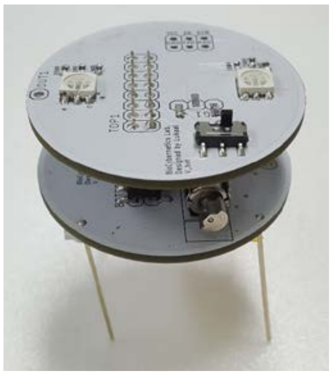
(c)

(d)

Fig. 3.8. 4th Hardware (a) PCB CAD of bottom board (b) Top and bottom view of bottom board (c) PCB CAD of top board (d) Top and bottom view of top board

In the 3rd Hardware, we changed to PCB of 2 layers shown in figure 3.8. We had come to know the limitations in the present structure even with the expanded PCB size. Therefore, we split the PCB to 2 layers and connected them using the 16-pin header connector. The Pins include power source and GPIO consisted of motor and LED control.

Lastly, we had to move the RGB LEDs position to the edge of the board because of motors. However, by doing so we couldn't distinguish between adjacent robot. Therefore, we moved the LEDs position to the inner board. This was possible due to the double board.



(a) (b) (c)

Fig. 3.9. 4th Hardware (a) Top-Left view (b) Front view (c) Right side view

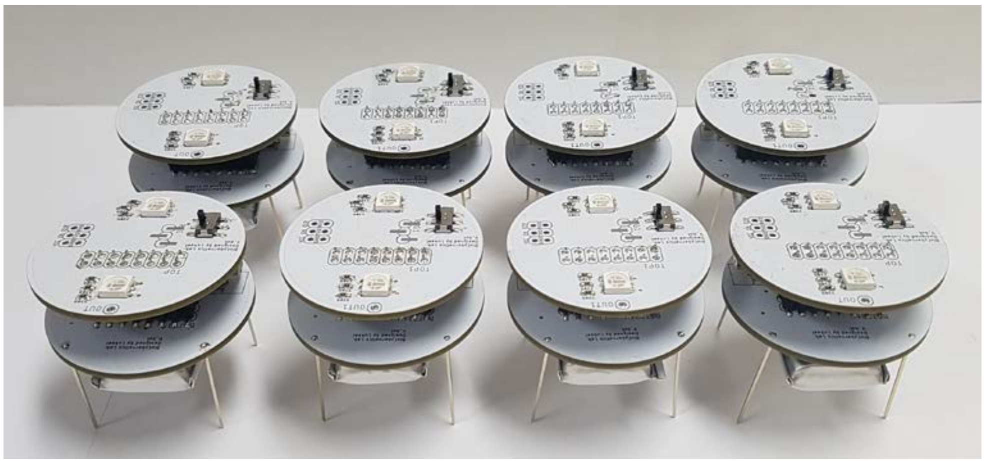


Fig. 3.10. Robot Swarms

3.2. Firmware

We developed our firmware using ESP-IDF, a Development Framework provided by Espressif. ESP-IDF is based on FreeRTOS: a real-time operating system kernel.

The code is divided into five parts, each consisting of FreeRTOS, flash memory, GPIO, Wi-Fi, and message parser. The main function conducts platform initialization and creates tasks. ‘platform_initialize’ sets the parameters required for using GPIO, Wi-Fi, and flash memory. This section will be explained in more detail in the sections that follow. After the initialization is completed, the robot turns on the two LEDs in blazing red to confirm that it is properly powered on. It then creates an ‘msg_control’ task that interprets incoming packets and a ‘tcp_conn’ task which is responsible for TCP / IP communications. Figure 3.11 is the flowchart of our firmware.

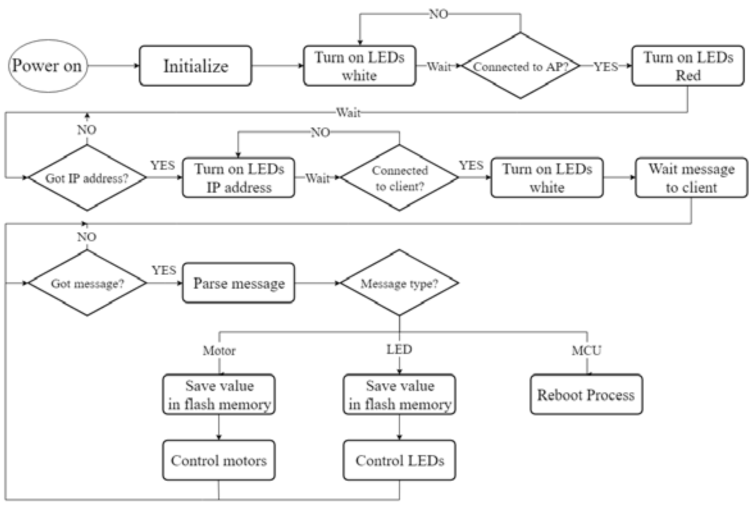


Fig. 3.11. Flowchart of Firmware

3.2.1. FreeRTOS

The FreeRTOS part schedules several tasks and exchanges data between them using interrupt queues. We have 3 tasks ‘tcp_conn’, ‘msg_control’ and ‘rxtx_data’. ‘tcp_conn’ task executes after connecting to AP. It creates TCP/IP socket and sets the hardware to server or client. ‘msg_control’ is the message parsing task and will be explained below. ‘rxtx_data’ is the task which receives and sends messages using TCP/IP and enqueues received messages in the interrupt queue. Interrupt queue is a kind of queue which generates an interrupt when enqueue.

3.2.2. GPIO

The GPIO part is responsible for the PWM output of the motor and LEDs. First, the MCU has two units with the same structure as the block diagram in Figure 3.12.

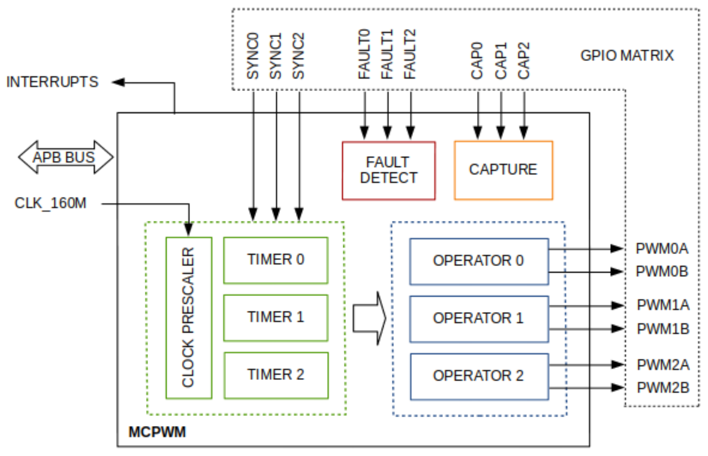


Fig. 3.12. Block diagram of MCPWM

Each unit has three Timers and Operators and each Operator can output two PWMs. Currently, two motors and two RGB LEDs are used in the robot, so two PWMs per motor and one PWM for each LED color is required for a total of 10 PWM outputs. Therefore, PWM0 and PWM1 of UNIT0 are set as motor outputs and PWM 0, 1 and 2 of UNIT1 are set for the 2 RGB LED outputs. This sets the necessary variables in the 'gpio_initialize' function as shown in the code below. As shown in the code below, we set PWM frequency to 500Hz and initialize duty cycle to 0.

```
//PWM Initialize
        pwm_config.frequency = 500;        //frequency = 500Hz,
        pwm_config.cmpr_a = 0;        //duty cycle of PWMxA = 0
        pwm_config.cmpr_b = 0;        //duty cycle of PWMxb = 0
        pwm_config.counter_mode = MCPWM_UP_COUNTER;
        pwm_config.duty_mode = MCPWM_DUTY_MODE_0;
        mcpwm_init(MCPWM_UNIT_0, MCPWM_TIMER_0, &pwm_config);
//Configure PWM0A & PWM0B with above settings
```

Also, the GPIO part controls the ADC using the battery indicator. ESP32 has 2 ADC drivers and each driver has 8,10 channels. However, ADC2 driver has some restrictions for application because of the Wi-Fi driver. Therefore, we can’t use the ADC2 driver.

3.2.3. Flash memory

The Flash memory part manages the nonvolatile memory in the chip. It manages the variables to be stored even if there is no power. In the initialize part, we allocated a new page in flash memory and set an initial value when no data is in flash memory. The initial value is determined by experimentation. The stored variables store the brightness information of the two LEDs and the motor output value. This is done so that we do not have to set it every time we turn the power off and back on. Also, each robot by nature has different weight distribution and leg structure. Therefore, each robot has different fine-tuned motor values to steer it in the correct orientation. We programmed these values in the robot’s firmware, so we do not have to fine-tune it every time.

3.2.4. Wi-Fi

The Wi-Fi part is responsible for AP/station configuration, TCP/IP network stack, and server/client configuration. Since the manufactured hardware is controlled through communication with the computer, it is difficult to communicate with a plurality of robots when it is set as an AP. Therefore, we set the robots as stations. Also, in order to distinguish the robots through their IP addresses we set the robots as servers so that each robot has a unique IP address. Then, we set the port number to 5000. The 'wifi_init_sta' function receives the SSID and password of the previously specified AP, stores it in the variable, and sets the necessary variables. This is done in ‘platform_initialize’. The 'tcp_conn' task sets the TCP/IP network stack through the 'create_tcp_server' function. At this time, the LEDs are turned off and rebooted when an error occurs and all the colors of RGB are turned on when the operation is successfully performed. This is to aid the user when the user needs to debug the system. When the initialization is completed, the AP assigns a IP address to the robot and LEDs are turned on accordingly to the IP address in a 6-bit binary number. This is useful to find the IP address without manually connecting to the AP or through serial communication. When the connection is completed a task called 'rxtx_data' is created and the data is received from this task.

3.2.5. Message parser

The message parser analyzes the packet received from 'rxtx_data'. When data enters the queue, an interrupt occurs. Then, the packet is separated into JSON-formatted packets through the JSON parser. This packet is analyzed through the 'msg_decode' function and the robot is operated according to the message. In the ‘msg_decode’ function, it has a ‘jsoneq’ function which compare JSON packet with tokens then returns the value of the corresponding token. After returning value it executes the adequate function.

3.2.6. Protocol

As a communication protocol, JSON (JavaScript Object Notation) protocol which is easy to write for both people and computers was used to control the color and brightness of LEDs and motor speeds. The protocol consists of the following components.

{"left": 0, "right": 0, "CW", "CCW", "hLed": 0, "tLed": 0,

"rhLed": 85, "ghLed": 85, "bhLed": 85, ”rtLed": 85, "gtLed": 85, "btLed": 85}

Name	**Description**	**Value Range**
Left	Left motor speed	-255~255
Right	Right motor speed	-255~255
The higher the number, the faster. Negative number means reverse. Save motor value (except 0) in flash memory.		
CW	Clockwise	None
Left motor turns on at flash value.		
CCW	Counter Clockwise	None
Right motor turns on at flash value.		
hLed	Head LED color	0~7
tLed	Tail LED color	0~7
0. Off 1. Blue 2. Green 3. Cyan(Blue+Green)	4. Red 5. Magenta(Red+Blue) 6. Yellow(Red+Green) 7. White(Red+Green+Blue)	
rhLed	Head Red LED intensity	0~255
The lower the number, the Brighter.		
ghLed	Head Green LED intensity	0~255
The lower the number, the Brighter.		
bhLed	Head Blue LED intensity	0~255
The lower the number, the Brighter.		
rtLed	Tail Red LED intensity	0~255
The lower the number, the Brighter.		
gtLed	Tail Green LED intensity	0~255
The lower the number, the Brighter.		
btLed	Tail Blue LED intensity	0~255
The lower the number, the Brighter.		

Table 3.2. Protocol table

4. Software

If the firmware's job is to take care of lower level operations such as transferring a command into an action, the software's job is to give the command. It can be looked at as the brains of the robots or it can be seen as the conductor. For our implementation, it is a mixture of both. The software looks at the entire scene using vision and gives commands to each individual robot such as 'go to a given location from current position' much like a conductor. The robot must then decide which motions it should take in order to get that given location, much like a brain. In the following parts we will discuss how vision is used to obtain the scene, how the robot moves and how the robot makes decisions. Lastly, using these features we will discuss certain applications that are viable for our robots.

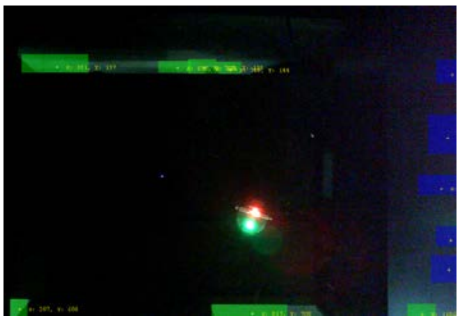
4.1. Vision

In order to control the robot, we need to acquire the robot's position and orientation. We do this by using computer vision with a camera and on-board LEDs. The primary task for vision is recognizing the head and tail LEDs of the robot. Once we know the coordinates of the head and tail LEDs, we can calculate the mid-point to obtain the position of the robot and the arc tangent to obtain the orientation of the robot. We must also be able to distinguish between several independent robots. This is a main issue in controlling swarm robots and is covered here as 'tracking'.

4.1.1. Pre-thresholding, 2-D median filter

We use the Blob Analysis block supported in MATLAB to calculate statistics for colored regions. The block returns quantities such as the centroid, bounding box, and blob count. Using this tool, we are able to obtain coordinates for the LEDs as shown in figure 4.1. A major problem for using LEDs is the glare on the camera lens and the tendency for colors to appear white when the intensity is large. Another problem is color distortion due to the background lighting or camera specs. On the left of figure 4.1, the red and green extend to the right of the robot. This can be misinterpreted as another robot or the center position can be altered. Also, due to external light some parts of the black background have been distorted to show some unwanted green or blue regions.

We solved this problem by applying pre-thresholding to set boundaries for the color regions and a 2-D median filter to get rid of noise. The result is a correct identification of the robot's head and tail as shown on the right of figure 4.1.


(a)

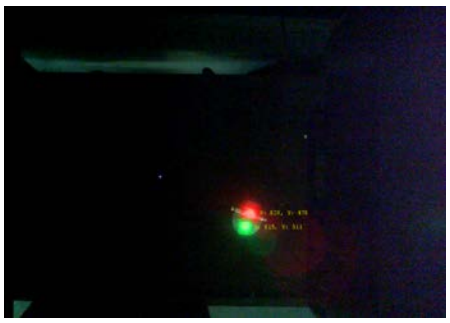
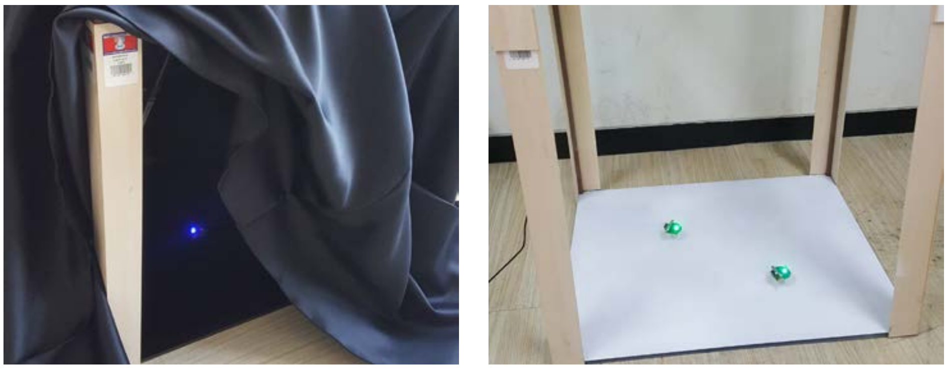

(b)

Fig 4.1. LED detection using computer vision. (a) Without pre-thresholding, median filter (b) With pre-thresholding, median filter

4.1.2. Effect of test bed, LED color

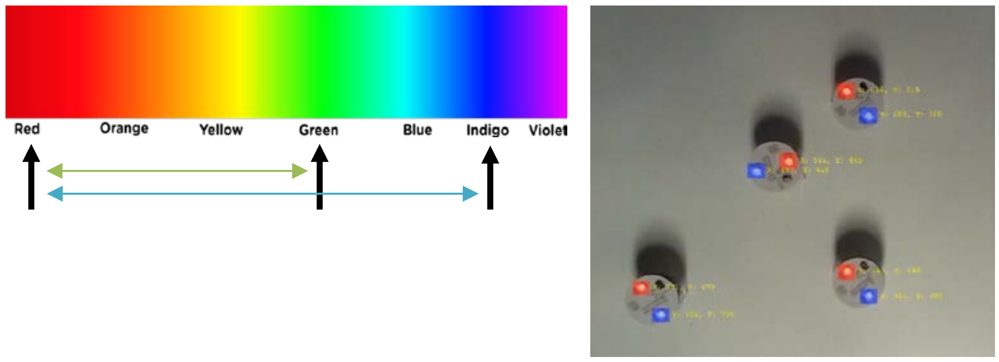
We also created a test bed. The platform on which the robot roams is a thick acrylic sheet with wooden pillars to sustain the blackout curtains. The blackout curtains are made of 100% polyester and are used in photo studios and laser experimental labs. By using this test bed shown on the left of figure 4.2, we could eliminate all excess light. However, we wanted to improve the system even further. Although we were able to get rid of all external light it was still impossible to eliminate the light generated from the LEDs itself from bouncing off the platform and walls. We found the best way to prevent this was actually to use a white background with much excess white light shining in on the platform. The excess light and white background actually had the effect of covering up any unwanted color spills. Also, all excess light loses any characteristics because the RGB values of the white background are already very high.



(a) (b)

Fig. 4.2. (a) Black test bed with blackout curtain (b) White test bed without curtain

We also changed the LED colorization to red and blue which is further apart in the color spectrum than red and green, thus easier to distinguish and identify. Finally, we were able to obtain the solid boundary blocks for the head and tail of the robot in figure 4.3.

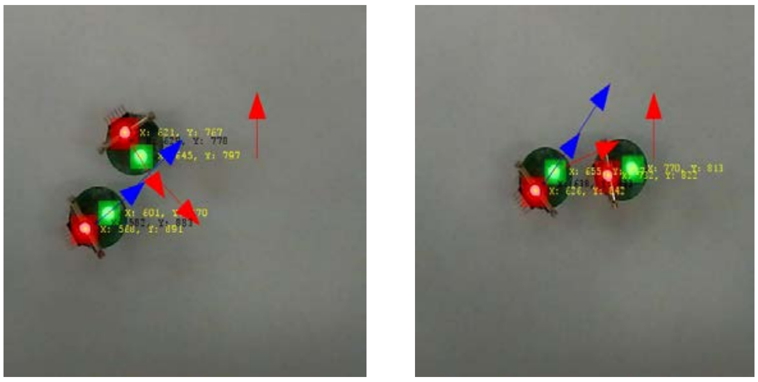


(a) (b)

Fig. 4.3. (a) Color spectrum (b) Final refinement of LED detection

4.1.3. Tracking

Tracking is very important in swarm robotics. Since there is a multitude of robots roaming around together, we need a method to keep track of and control each robot specifically. In our case we need to find the correct LED coordinates that correspond to the individual robot i.e. we need to match the LEDs to the robots. To do this we use a simple Nearest Neighbor (kNN) matching algorithm: a greedy algorithm which goes through the potential matches and selects the closest unmatched option each time. The propensity score[4], which is the criteria for closeness, is defined as the Euclidean distance from the previous coordinate of a robot to the newly obtained LED coordinate. Since the nearest neighbor algorithm simply gives the 'nearest' neighbor, one can end up with a very bad match if the nearest neighbor is far away. In our case if one of the LEDs is undetected by the camera then there can be a chain reaction of catastrophic failure of tracking. Therefore, we set 'tolerance levels'[4] (i.e. upper limits) to determine how far our matching algorithm should go in search of the nearest neighbor. The tolerance levels are determined by the radius of the robot. Doing so also prevents the case of 'coordinate stealing' where two robots come too close and the algorithm mistakenly assigns the coordinates of one robot to the other robot as seen in figure 4.4.



(a) (b)

Fig. 4.4. Coordinate stealing (a) Robots come too close. (b) Blue robot steals coordinate of red robot.

The red and blue arrows represent the orientation of the robot and the trajectory that it is following. In the figure on the left each robot has its own set of arrows distinguishable by color. On the right the blue and red arrows overlap as it is stolen by the left robot.

4.2. Robot locomotion

How the robot is going to move is a major aspect in any robot design. The designer must consider the application in which the robot will be used and leverage the pros and cons of many possible implementations. For swarm robots an important factor is cost. Since there is a multitude of robots that need to be built each individual robot must be cost friendly. Another factor is maneuverability. Since swarm robots interact with many other robots it is always placed in an environment with many obstacles being its robot counterparts themselves. We chose a means of locomotion that best considers these factors. Our robot is able to maneuver itself simply by using two vibration motors on each side. The two vibration motors allow the robot to change its direction using the difference in intensity of the two motors.

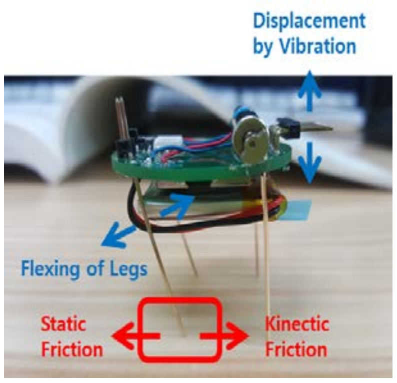


Fig. 4.5. Leg structure’s effect on locomotion

The leg structure is a very important factor for determining the robot’s locomotion. Small changes to the orientation of the legs can result in completely different movements for the same motor values. Through various experimentations we found that the leg structure in figure 4.5 is best suited for a stable movement across smooth surfaces. The inward flexing of the legs takes advantage of the difference in static friction and kinetic friction that apply on the feet of the robot. This results in an overall forward motion with changes in orientation as motor values are changed. Similar approaches have been found in Kilobots[5] and Planar Micro Robots[6] where two centripetal-force actuators are used to drive the legs using the slip-stick locomotion explained above (difference in static and kinetic friction). However, our approach of flexing the legs gives an additional improvement in speed i.e. the difference in friction is enhanced and the robot is able to move faster.

4.2.1. Problem of inconsistency of motor values

One major setback in using the vibration motors for controlling the robot is the inconsistency of locomotion displayed due to various external factors. By controlling the intensity of vibration for the two motors, the robot should be able to move in a continuous range from clockwise rotation, to straight forward, to counterclockwise rotation. However, this is not the case in most environments. The robot's movement has a large error when it comes to slight inconsistencies of the surface. External factors such as small slopes or bumps can occur and alter the robot's motion significantly. Therefore, using the vibration motor values as those of wheels is a very difficult task.

4.2.2. Rotational motion to linear motion

We solve this problem by greatly simplifying the robot's locomotion from a continuous range of orientation to a discrete binary range of rotation. Through experimentation we found that although it is very difficult to tune the motors for the robot to move in a specific orientation it is relatively easy to flex the legs so that the robot rotates either clockwise or counterclockwise given either a left motor value or a right motor value. By flexing the legs in an appropriate manner, we can adjust the rotational axis that the rotational motion is made. Also, by alternating between clockwise and counterclockwise motion and by shifting the rotational axis along with it, we can transfer the rotational motion into a linear motion as in figure 4.6.

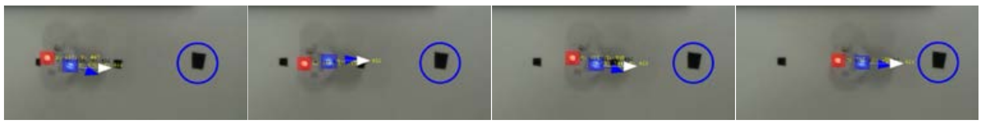


Fig. 4.6. Transferring rotational motion to linear motion

Take note that the blue arrow representing the orientation of the robot shifts from side to side of the white arrow representing the trajectory that the robot is following. In this manner we use a control algorithm to generate a desired path to reach its goal point and use alternating rotational motion to follow that path.

4.3. Control algorithm

Once the foundations for the robot are set all that is left is to control it, program its inner workings and give it life. Selecting the appropriate control algorithm is largely responsible for determining how well the robot performs. Also, different algorithms may display different behavioral personalities. To control the robot, we look at two algorithms. The POSQ extend function proposed by Palmieri et al[7] and the Dynamic Window Approach algorithm proposed by Dieter et al[8]. For swarm robots, collision avoidance is also a very important factor when it comes to control algorithms. We will discuss some approaches we took in addition to these two algorithms and the modifications we made to make the algorithms more suitable for our applications.

4.3.1. POSQ Extend function

The first control algorithm, POSQ, was originally designed as an extend function for the RRT (rapidly-exploring random trees) algorithm. It generates a trajectory connecting any given pair of poses (thus giving it its name, POSQ). It is based on the kinematic model of a non-holonomic wheeled mobile robot. A robot is holonomic with respect to N dimensions, if it is capable of moving in any direction in any of those N physical dimensions available to it. If it is non-holonomic, it is restricted in which directions it can move in. The method makes a Cartesian-to-polar coordinates transform to describe the kinematics using an open loop model and feedback law and computes closed-loop forward simulations based on the kinematic model of the robot and enables the planner to efficiently generate smooth and feasible paths. There paths were suitable for our robot which was unable to make sharp turns and based on the rotational radius of each robot we were able to generate end-to-end robot-specific trajectories using this algorithm.

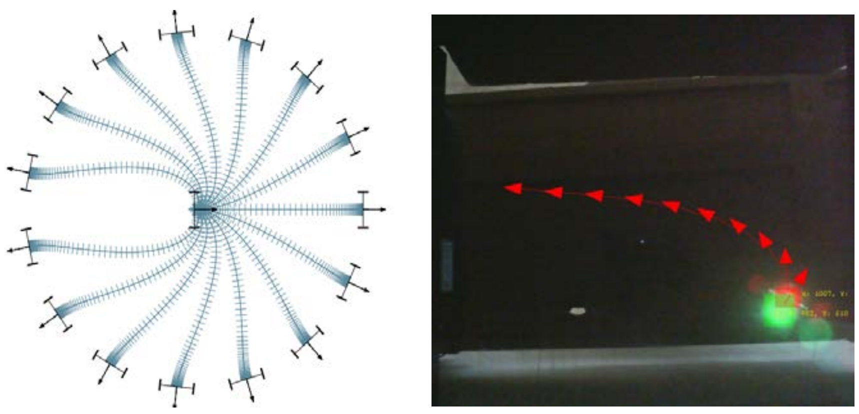


Fig. 4.7. Trajectory generation using POSQ extend function

However, the problem with this algorithm was using it as an extend function for the RRT algorithm. Collision avoidance by means of RRT was not suitable for our application. The RRT requires too much computational time and is suited for environments where obstacles are stationary. In our application obstacles are constantly moving and the robot is also prone to position and orientation errors. This calls for constant running of the full cycle of RRT thus greatly reducing the control speed. So, we only use the extend function for generating trajectories to a given position and apply collision avoidance in a different manner.

4.3.2. POSQ Extend function with collision avoidance vector

Since we could not use RRT, we needed a new method to avoid collision. The trajectory generation with POSQ was good and was able to constantly regenerate with a fast control speed. Therefore, we modified the POSQ trajectory by adding it with an additional vector called the ‘collision avoidance vector’. This vector has a magnitude of inverse the distance to the obstacle and a direction opposite of the obstacle.

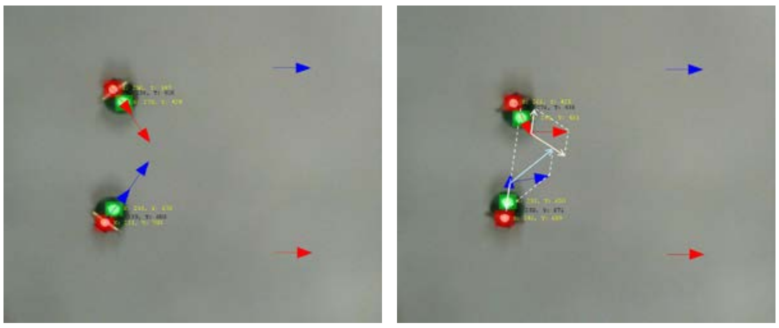


Fig. 4.8. Collision avoidance using POSQ extend function

The left of figure 4.8 shows the original POSQ trajectory before collision avoidance kicks in and the right shows the POSQ trajectory added with the collision avoidance vector.

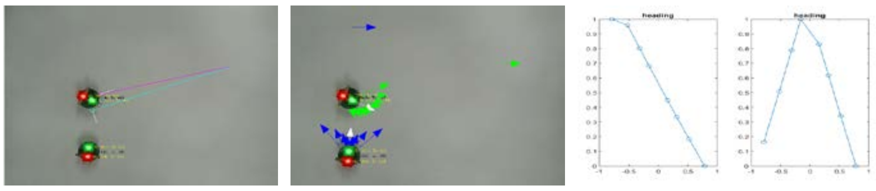
Although this method prevents the robots from colliding, the robots also tend to fall into a cycle of collision i.e. the collision avoidance kicks in regardless of the goal. This leads to robots being unable to reach their goals if the goals are close to each other.

4.3.3. DWA*

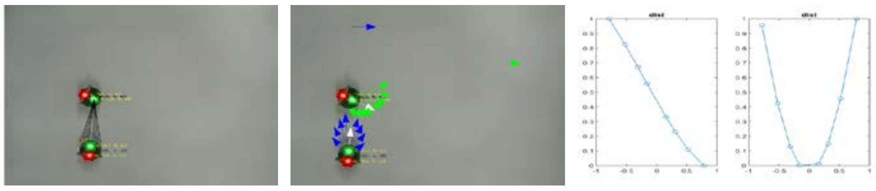
Our second control algorithm is a modified version of the Dynamic Window Approach. The original DWA was suited for two wheeled robots with a solid kinematic model. However our vibration robot has no existing kinematic model and velocities are unpredictable. Therefore we proposed a slightly different approach and called it DWA*. We reduce the search space to one dynamic window which only consists of the angles reachable within a short interval. In our case we consider 8 angles (θ = $-\pi/4$, $-\pi/6$, $-\pi/10$, $-\pi/20$, $\pi/20$, $\pi/10$, $\pi/6$, $\pi/4$) in respect to the robot's orientation. Among these angles, the best trajectory is chosen by maximizing an objective function.

$$G(\theta)=\alpha\times \text{heading}(\theta)+\beta\times \text{distance}(\theta)+\gamma\times \text{momentum}(\theta) \qquad \text{(eq. 4.1)}$$

The objective function consists of 3 components: heading, distance and momentum. Each component is weighted with a parameter that determines the personality of the robot. Through experimentation we discovered that as the number of robots increased, β should be increased in respect to α (α=1, β=1.2 for 8 robots). γ is kept at a small value (γ=0.1). Figure 4.9 shows



(a)



(b)

Fig. 4.9. (a) shows the calculation method of the heading term. The calculated heading values for each angle are represented as the length of the arrows. The values are also shown graphically. We do the same for distance and are depicted in (b).

how the heading and distance components are calculated. The heading measures the alignment of the robot with the goal direction. It is calculated by comparing which angle has the smallest angular difference from the goal i.e. heading(θ) = $\pi - \phi$, where ϕ is the angle of the target point relative to the robot's heading direction when θ is selected. The distance measures how far the robot is from an obstacle. It is calculated by comparing which angle is furthest from the obstacle i.e. we measure the Euclidian distance of each angle from a predicted point that the obstacle is to move to. We are able to predict this point because we know each robot’s heading direction due to tracking. Finally, the momentum term reduces fluctuation of the robot. When the heading and distance terms are similar to each other it is difficult to choose the optimum angle. The robot therefore may fall into a randomly fluctuating state where it goes back and forth between angles. In this case momentum is calculated by giving the previous selected angle an additional score thus giving the robot a helping hand in escaping this fluctuating state.

Comparing it to the original DWA, the heading is the same and the distance was originally calculated by whether an obstacle existed in the selected velocity. However, we do not consider velocity and the robot should predict a collision beforehand using the predicted point because the obstacles are moving and there exists lag due to low frame rates for vision and lack of agility due to the vibration locomotion. Finally, there was originally no momentum, rather a velocity term. This was needed due to the rotational motion of the robot. In order for the rotational motion to transfer smoothly into a linear motion we found that reducing random fluctuation helped.

4.3.4. Comparison

Table 4.1 shows the pros and cons of POSQ and DWA*. Although the POSQ extend function is able to generate entire trajectories from start to finish it does not consider collision avoidance. Therefore an additional collision avoidance vector must be added but can lead to the robots falling into a cycle of avoidance. It is possible to think of better ways to incorporate collision avoidance into the POSQ extend function but even the generation of entire trajectories is actually wasted in our application since the robot’s trajectories require constant regeneration. The POSQ extend function’s best merit is that it considers the pose of the robot. This can prove to be useful in many applications. However for now we are only seeking to arrive at a target position and do not have any need for the end pose.

Our proposed method, DWA* does not consider pose but evaluates the best trajectory in real time leaving none of its procedures to waste. Also the leveraging between heading and distance allows collision avoidance to come naturally. However, fine tuning of the parameters are required and this can vary according to the environment. Regardless, it is not a major problem in our fixed setting and small distortions such as slopes and bumps are overcome without needing dynamic tuning.

By comparing POSQ and DWA* we decided that since we do not need to consider the final pose for our application DWA* was the best choice.

	POSQ	**DWA***
Pros	Generate entire trajectory. Pose is considered.	Real time evaluation of best trajectory. Collision avoidance comes naturally.
Cons	Robots fall into a cycle of avoidance.	Pose is not considered. Finetuning of parameters is required.

Table 4.1. Comparison between POSQ extend function and DWA*

4.4. Formation

Now that we are able to see and keep track of our robots, move our robots in a refined way and control them as a whole without bumping into each other we are able to apply them to various applications. In this paper, we demonstrate the swarm robots conducting formations from randomly initial positions. We also propose some other applications that might be applicable in future works.

Formation consists of giving the swarm a formation to conduct as a whole, distributing which robot should go to which position and controlling each robot. Munkras algorithm was used to optimize the distribution of goals[9][10].

The following 6-step algorithm is a modified form of the original Munkres' Assignment Algorithm (sometimes referred to as the Hungarian Algorithm) and distributes m jobs to n workers each with an individual cost. For our application we consider the jobs to be the goals, the workers to be the robots and the cost to be the Euclidian distance from the goals to the robots.

Step 0: Create an nxm matrix called the cost matrix in which each element represents the cost of assigning one of n workers to one of m jobs. Rotate the matrix so that there are at least as many columns as rows and let k=min(n,m).

Step 1: For each row of the matrix, find the smallest element and subtract it from every element in its row. Go to Step 2.

Step 2: Find a zero (Z) in the resulting matrix. If there is no starred zero in its row or column, star Z. Repeat for each element in the matrix. Go to Step 3.

Step 3: Cover each column containing a starred zero. If K columns are covered, the starred zeros describe a complete set of unique assignments. In this case, Go to DONE, otherwise, Go to Step 4.

Step 4: Find a non-covered zero and prime it. If there is no starred zero in the row containing this primed zero, Go to Step 5. Otherwise, cover this row and uncover the column containing the starred zero. Continue in this manner until there are no uncovered zeros left. Save the smallest uncovered value and Go to Step 6.

Step 5: Construct a series of alternating primed and starred zeros as follows. Let Z0 represent the uncovered primed zero found in Step 4. Let Z1 denote the starred zero in the column of Z0 (if any). Let Z2 denote the primed zero in the row of Z1 (there will always be one). Continue until the series terminates at a primed zero that has no starred zero in its column. Unstar each starred zero of the series, star each primed zero of the series, erase all primes and uncover every line in the matrix. Return to Step 3.

Step 6: Add the value found in Step 4 to every element of each covered row, and subtract it from every element of each uncovered column. Return to Step 4 without altering any stars, primes, or covered lines.

DONE: Assignment pairs are indicated by the positions of the starred zeros in the cost matrix. If C(i,j) is a starred zero, then the element associated with row i is assigned to the element associated with column j.

Fig. 4.10. modified Munkres' Assignment Algorithm

5. Result and Discussion

After multiple versions of hardware and refinement of software we were able to implement smooth formations using our swarm robots.

5.1 Result

We can see successful formations using the swarm robots in figure 5.1. The left is the initial randomly scattered robots and the right is the formations that the robots conducted.

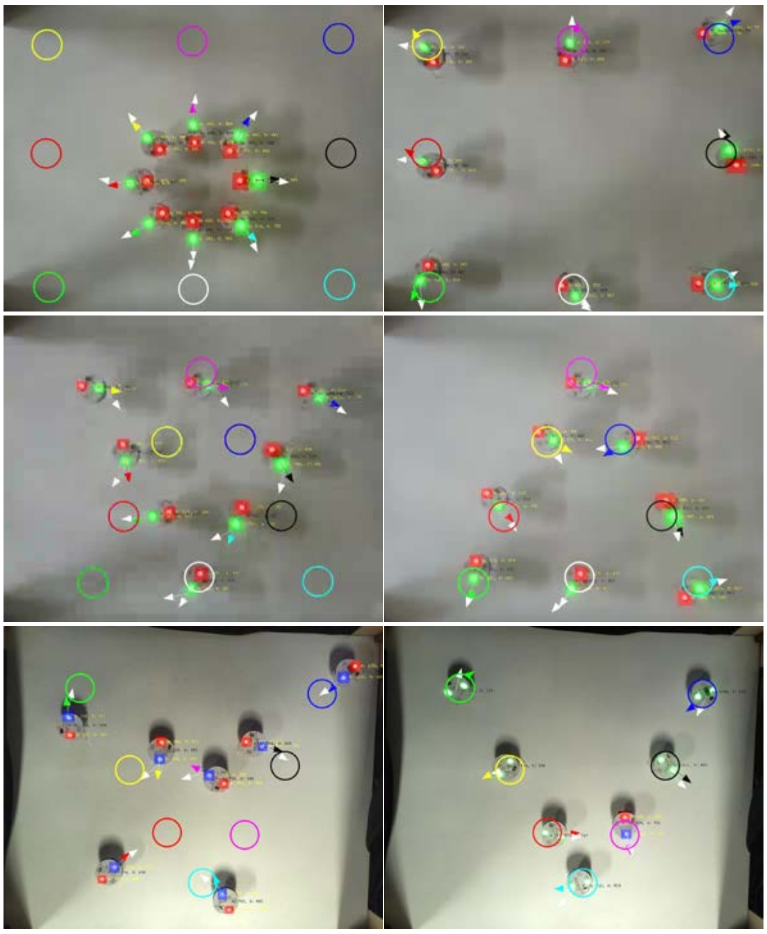


Fig. 5.1. Formation of vibration swarm robots

5.2 Discussion

Formation is a product of vision, locomotion and control. By analyzing the performance of formation, we can evaluate the performance of our system. During formation, sometimes a robot will be stuck in its tracks. This can be due to many issues. Firstly, vision might have failed to identify or track the robot. Although we have a pretty solid vision system it still sometimes loses its target when there is a disturbance in light. Also, the robot might have a sudden burst of speed due to the discontinuity of the surface. If the burst is faster than the frame speed, then vision might lose the robot while tracking. Finally, the robot might simply meet a disturbance in the surface and its legs might get stuck to the oddity. For vision, we propose that other methods of identification (other than LEDs) can be used such as QR codes or R-CNNs. To solve the tracking issue, we can add an emergency reset when the system detects such an event. Finally, for the sticking of legs we can give the motors a spike of intensity allowing it to break free from the odd surface. By refining our robot further using these methods we can obtain a more robust system.

5.3 Future Works

For future works we plan to add an OTA download feature to the firmware, improve charging method and perform multiple robot control. The OTA download will be used to upload firmware concurrently to multiple swarm robots using Wi-Fi. It can reduce the time when we need to modify firmware. We are planning to implement charging by means of robot to robot for the charging method.

We also would like to implement a division of labor algorithm and some animal mimicry. Instead of specifically distributing the goals using the Munkres algorithm globally, a division of labor algorithm would be able to distribute goals locally by seeing which goals have less robots approaching it and dispersing to achieve those goals. In advancement to formation which is inspired by migratory birds, we plan on implementing other animal mimicry applications such as shepherding (behavior of shepherd dogs and sheep), and schooling (behavior of fish schools).

6. Conclusion

Designing and implementing a biologically inspired robot is all about making it able to ‘live’ in its environment. For hardware, we went through the robot’s motors (how the robot moves in its environment), structure (how the robot is formed to best suit to its environment), and firmware (how the robot thinks and processes information obtained from its environment).

For hardware, we made 5 robots and they have been supplemented. First, we were testing vibration motor control and made 1st Hardware. We selected MCU and added necessary parts to control. After found improvements in 1st Hardware, we fixed it and add feature in 2nd Hardware. In 2nd Hardware we found critical problems and we made 3rd Hardware that modified it. Lastly, we made 4th Hardware that changed PCB in 2 layers and added some feature. After made hardware, we developed firmware using ESP-IDF. We created and scheduled tasks in firmware using FreeRTOS and initialized Wi-Fi, GPIO, flash memory. Also, we selected JSON format to message protocol.

For software, we went through vision (how the robots sees and interprets its environment), locomotion (how the robot maneuvers around its environment), and control (how the robot manipulates and is manipulated by its environment). We used various methods to improve each aspect of our system. For vision we used pre-thresholding, 2-D median filtering, and optimum colorization of the test bed and LEDs to improve LED detection and tracking. We transferred rotational motion to linear motion for robust locomotion. We proposed a modified version of a well-known algorithm, DWA*, to better suit our application. Finally, we put this all together to implement a fundamental application that tested our robots in all-round aspects.

Reference

[1] James MacDonald, (2017), How Do Fish Schools Work? [ONLINE]. Available at: https://daily.jstor.org/how-do-fish-schools-work/ [Accessed 12 December 2018].

[2] Veneeta Prasad, (2018), V-formation of migratory birds [ONLINE]. Available at: https://www.lifelessons.co.nz/ [Accessed 12 December 2018].

[3] Espressif, (2018), Block diagram of MCPWM [ONLINE]. Available at: https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/mcpwm.html [Accessed 12 December 2018].

[4] Caliendo and Kopeinig, Some Practical Guidance for the Implementation of Propensity Score Matching, IZA Discussion Paper No. 1588, May 2005.

[5] Michael Rubenstein, Christian Ahler, and Radhika Nagpal, Kilobot: A Low Cost Scalable Robot System for Collective Behaviors, IEEE Intl. Conf on Robotics and Automation (ICRA), 2012.

[6] P. Vartholomeos and E. Papadopoulos, Analysis, design and control of a planar micro-robot driven by two centripetal-force actuators, ICRA, 2006.

[7] L. Palmieri and K. O. Arras, "Efficient and Smooth RRT Motion Planning Using a Novel Extend Function for Wheeled Mobile Robots", IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 205-211, 2014.

[8] Dieter, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance", IEEE Robot. Autom. Mag., vol. 4, no. 1, pp. 23-33, Mar. 1997.

[9] James Munkres, Algorithms for Assignment and Transportation Problems, Journal of the Society for Industrial and Applied Mathematics Volume 5, Number 1, March, 1957.

[10] F. Burgeois and J.-C. Lasalle, An extension of the Munkres algorithm for the assignment problem to rectangular matrices, Communications of the ACM, 142302-806, 1971.

국 문 요 약

진동로봇을 이용한 다양한 자연모방 군집행동

자연에서, 동물들은 살아남기 위하여 다양한 방법으로 환경에 적응하고 진화해 왔다. 이 중에서 본 연구진은 개미, 물고기 떼, 철새 떼 등이 보여주는 특이한 군집 행동에 관심을 가지게 되었다. 각 객체는 일정한 규칙을 따르며 개별적인 행동을 하지만 이 객체들이 모여서 집단적인 행동으로 나타나며 이를 '군집 행동'이라고 한다. 이 행동을 모사함으로써, 본 연구진은 복잡한 문제에 대한 새로운 해답을 제시하고자 한다. 또한, 본 연구진은 이를 통해 자연과 동물들의 행동을 쉽게 이해할 수 있도록 하고자 한다.

본 눈문에서는 이러한 동물들의 군집 행동을 로봇을 통해 모사하고자 하였고 이를 위해 진동 로봇을 설계하고 이를 제어할 알고리즘을 제시하였다. 본 연구진은 군집 로봇을 설계하기 위한 3가지 조건을 제시하였고 여러 차례의 실험과 시행착오를 통해 위 조건에 맞는 로봇을 제작하였다. 또한, 컴퓨터 비전을 통해 로봇의 위치를 파악하고 기존의 알려진 알고리즘을 수정하여 진동 로봇에 적합한 제어 알고리즘을 설계하였다.

이 시스템을 검증하기 위해 군집 행동 중 대형 이루기에 관한 실험을 진행하였고 이를 통해 다른 군집 행동을 모사하는 것에 대한 가능성을 제시하였다.

핵심되는 말 : 군집 행동, 진동 제어, 군집 로봇, 생체 모방, 충돌 회피, 목표 지점 도달, 대형, 컴퓨터 비전, 제어 알고리즘