EFFECTIVENESS OF CLOSED-LOOP INVERSE-KINEMATIC LANDMARK NAVIGATION METHOD ON DRIFTING AND HEADING ERRORS OF IMPRECISE MULTI-LEGGED ROBOTS

By

Thong Quang Nguyen

THESIS

Submitted in partial fulfillment of the requirements for the degree of

Master of Science in Mechanical Engineering

at The University of Texas at Arlington

May 2021

Arlington, Texas

Acknowledgement

I would like to thank Dr. Alan Bowling for his continuous support and guidance over last two years of my master's degree at University of Texas at Arlington.

I would like to thank Dr. Kamesh Subbarao and Dr. Shuo (Linda) Wang for evaluating my work as committee members.

I would like to thank all my friends in RBDSL laboratory team for helping me and supporting me throughout my research.

I would like to thank my families for all the supports they have given to me. Without them, I would not have been able to pursue my master's degree.

Abstract

EFFECTIVENESS OF CLOSED-LOOP INVERSE-KINEMATIC LANDMARK NAVIGATION METHOD ON DRIFTING AND HEADING ERRORS OF SLOPPY MULTI-LEGGED ROBOTS

Thong Quang Nguyen, MS

The University of Texas at Arlington, 2021

Supervising Professor: Dr. Alan Bowling

This work addresses the effectiveness of a closed-loop inverse-kinematic control methodology on alleviating drifting and heading errors in the walking and turning of a hexapod robot. Most work in this area focuses on stability, hardware design, and control. This work is aimed at controlling imprecise hardware on unknown structured environments. The proposed controller uses a camera and an ultrasonic sensor as sensory feedback. It also uses the well-known tripod gait for locomotion, implemented with the closed-loop inverse-kinematic method. Experimental results show the effectiveness of this approach in walking and turning the hexapod on an unknown surface.

Table of Contents

List of Illustrations

Chapter 1

1.1 Introduction

The navigability of an autonomous multi-legged robot is an essential element of its capabilities. In automation, landmark navigation is commonly used for planning footsteps from the present position, [1]–[4]. Experiencing errors in moving is inevitable. Open-loop control is commonly used for multi-legged robots [5]–[9]. It normally leads to drift and heading errors, which are common for a mobile robot. In the task of alleviating these errors, closed-loop landmark navigation plays an important role. It essentially creates a feedback system by sensing landmarks in the environment. Then, the actual position can be computed, and compared with expected ones in motion planning to fix the errors. In this paper, an imprecise hexapod is automatically controlled in moving from a random point to a desired point. Closed-loop control is applied to fix the drifting and heading errors caused while walking and turning. A camera and ultrasonic sensor are attached to collect landmark data for the method, and the tripod gait is used as the method of locomotion. The effectiveness of the method will be assessed by the results of experiments.

Common approaches to navigation are classified as landmark navigation [1]–[4], odometry [10]–[12], inertial navigation [13]–[15]. Besides, there are several advance methods, such as vision-based [16]–[18], reinforcement learning [19]–[22]. However, these approaches usually face with heading and drifting errors [11], [23]. There are few efforts on fixing this problem on each method. In [24], Antonelli addressed a calibration method to alleviate the errors with a new odometry calibration of differential-drive mobile robots. In [25], Allan Variance method was addressed on gyroscope random errors, including drifting errors. In 2006, Bowling

had an approach with a simple method for measuring heading and drifting errors [26]. His method simply uses closed-loop inverse kinematic theory to correct the heading and drifting errors caused by moving. A camera, distance sensor, and compass are used to create a closed-loop system. By using feedback from camera, sensor and compass, a correction can be computed for the next step, based on expected and actual drift from previous step.

Beyond navigation approaches, stability has also gained attention, in the task of alleviating errors [27]–[29]. Enhancements in the hardware and control have generally improved the stability of hexapods, such as adaptive gait [30], [31], force-based margin [32], contact-point gait [33], etc. Additionally, these methods are commonly studied on unstructured terrain [30], [33]–[39].

This work examines how well the approach outlined in [26] can work on an imprecise hexapod in moving from an arbitrary initial point to a target point. A camera and an ultrasonic sensor are used as sensory feedback. The hardware design will be discussed next, followed by the control methodology, and experimental results showing the effectiveness of this approach. The experiments are implemented on structured, even terrain.

1.2 Hexapod

1.2.1 Hexapod Design

The hexapod design chosen for this study is imprecise and includes a camera and ultrasonic sensor. There are several different designs of hexapods that have been developed. The most typical one is similar to Tarry I, which has three DOF on each leg and can move vertically and horizontally [42]. The horizontal movement is not necessarily required in this experiment.

The hexapod only needs the ability of moving forward and turning. To have an imprecise design and instability of movement, a DOF can be neglected in each leg. According to the neglect, the best design for hexapod in this experiment is the HexCrawler design [43], shown in Figure 2. A single leg of the hexapod is controlled by only two servos. The Femur and Tibia, shown in Figure 1, are connected by a fixed joint, 90° in this case. A camera and ultrasonic are attached to its head.

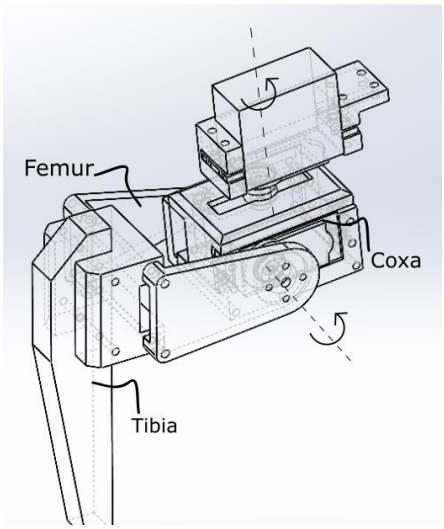


Figure 1. The two DOF actuated by two position-controlled servos.

Figure 2. HexCrawler.

1.2.2 Hardware

The hardware used to build the hexapod are a microcontroller, a camera, a printed circuit board (PCB), a battery, a voltage converter, 12 servos, an ultrasonic sensor, and 3D printed parts. The microcontroller is used to control the servos and sensor, and to receive data from the sensors. The microcontroller used in this experiment is an Elegoo mega 2560 R3, shown in Figure 3. It is compatible with Arduino IDE. The purpose of choosing the Arduino is its ready-to-use structure, its low cost and its open-source architecture. A user can easily create applications with minimal knowledge of electrical engineering.

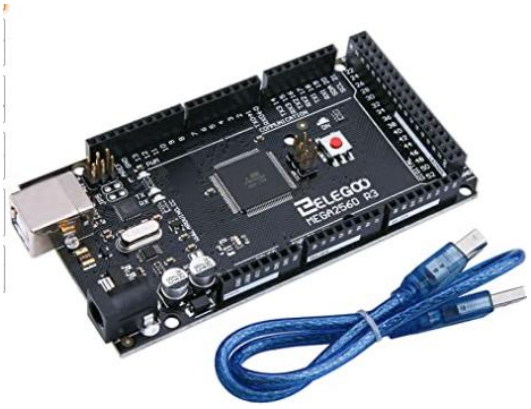


Figure 3. Elegoo mega 2560 R3.

The camera detects the targeted object and transmits the center coordinates of the object to the microcontroller. The criterion for the camera is that it is affordable and able to detect simple color objects. Thus, the camera used in this experiment is the Pixy 2 camera, shown in Figure 4. This camera can learn an object by memorizing the color texture of the object. It has 60° camera, and its frame width is 316 [27]. It uses In Circuit Serial Programming (ICSP) head to communicate with the Elegoo. It also has its own library on Arduino for use.

Figure 4. Pixy 2.

Due to a large number of wires and current use from servos, sensor, microcontroller and camera, a printed circuit board is needed to help organizing the wires, and power input from battery. It is designed on EasyEDA tool and printed by Jlcpcb website. The PCB in this experiment is shown in Figure 5.

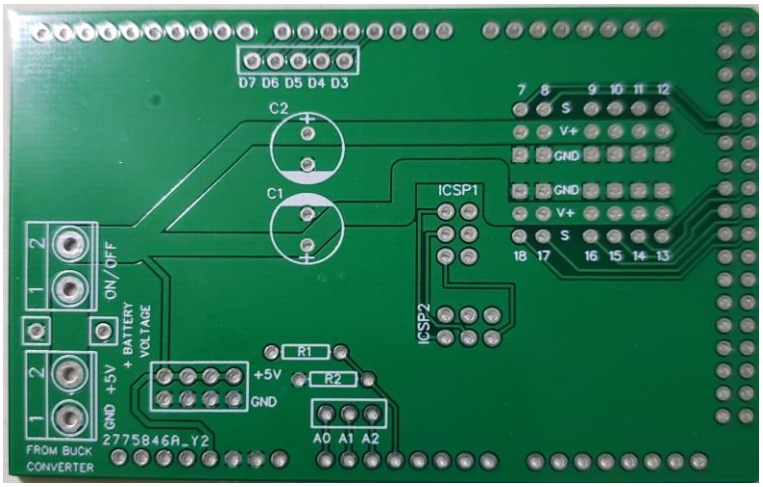


Figure 5. PCB.

The battery used is a 12V Tattu Lipo Battery with 1300 mAh, which has more than enough capability to supply current to other hardware, shown in Figure 6. However, other hardware can only accept 5V as the maximum input voltage, so a DC converter is needed.

Figure 6. Battery and DC converter.

All of the equipment used has voltage input of 5V. Thus, a DC converter is needed to convert 12V from the battery into 5V for the use of equipment. The DC converter used is Diymore Synchronous Buck Converter, shown in Figure 7. It can convert from 8-60V to 1-36V, with maximum current of 15A.

Figure 7. Diymore Synchronous Buck Converter.

The legs are controlled by servos. The servos used in here are MG996R servo with maximum torque of 9.4kg-cm at 4.8V, which is technically enough to control 3D printed legs without lack of torque, shown in Figure 8. Its maximum current is 3A. With 12 servos, it would

exceed the maximum current that the battery can supply. However, with low-weight legs, the servo will never need that much current.

Figure 8. MG996 servo.

The ultrasonic sensor is used to detect the presence and distance of the targeted object. HC-SR04 sensor is used and attached at the front of the hexapod along with the pixy camera, shown in Figure 9. Its detection zone is from 2cm to around 500cm. Its effectual angle is 15°. It also requires a 5V power source and the current used is less than 2mA.

Figure 9. Ultrasonic sensor.

The hexapod parts were 3D printed. The reason for choosing 3D printing methods is that they take less time and are more flexible for building prototypes than other methods. The 3D computer aided design (CAD) model is shown in Figure 10.

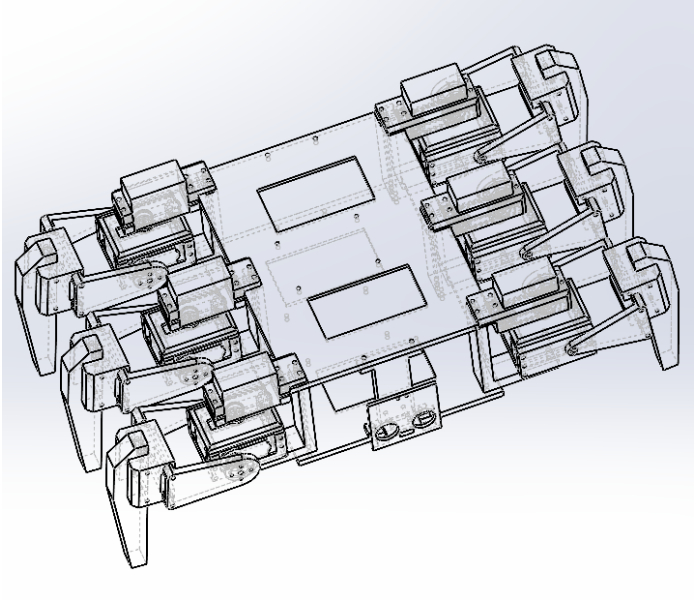


Figure 10. Hexapod Solidwork 3D model.

1.2.3 Software and Control

A hexapod is a complicated robot. It has 6 legs, which creates a large number of degrees of freedom (DOF). An ineffective control can cause some significant errors while moving. The hexapod can achieve locomotion using four different basic gaits: Tripod gait, Wave gait, Tetrapod gait and Ripple gait [40]. The wave and ripple gait have slower speed and create more stability than the tripod gait, which might create smaller errors while moving [41]. The tetrapod gait is a more complex motion, but its stability is higher than tripod. The tripod gait is used in this work for its speed of locomotion. See appendix I for the tripod gait and appendix II for software used for the experiment.

Chapter 2

2.1 Problem

The drifting and heading phenomena are common issues of navigability of multi-legged robots. These two phenomena are caused by the inaccuracy and instability of the system. When an open-loop control system is used, shown in Figure 11, the actual position of the hexapod is not frequently updated. This might cause a flaw between actual and desired positions.

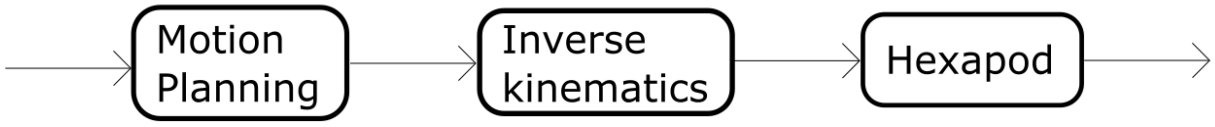


Figure 11. Open-loop Control

The drifting error is measured by the distance that the robot drifts off from the expected trajectory and the heading error is the angle that the robot shifts from the expected trajectory. Figures 12 and 13 show examples of these types of errors. Figure 12 shows that drifting errors always exist for every step of the hexapod in walking forward. Figure 13 shows the heading errors in turning of the hexapod. The distance in the figure is the distance between the target object and the head of hexapod.

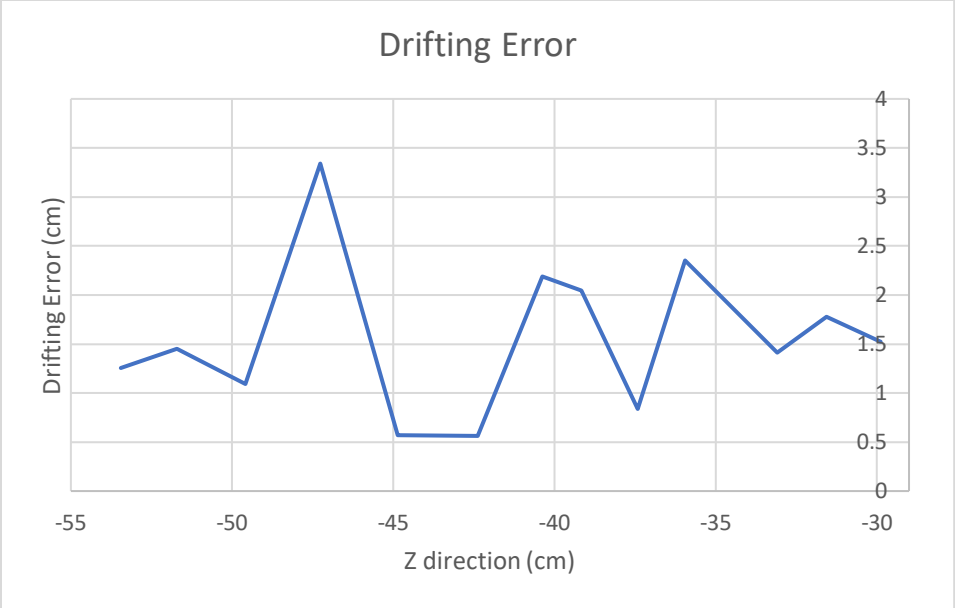


Figure 12. Drifting phenomenon

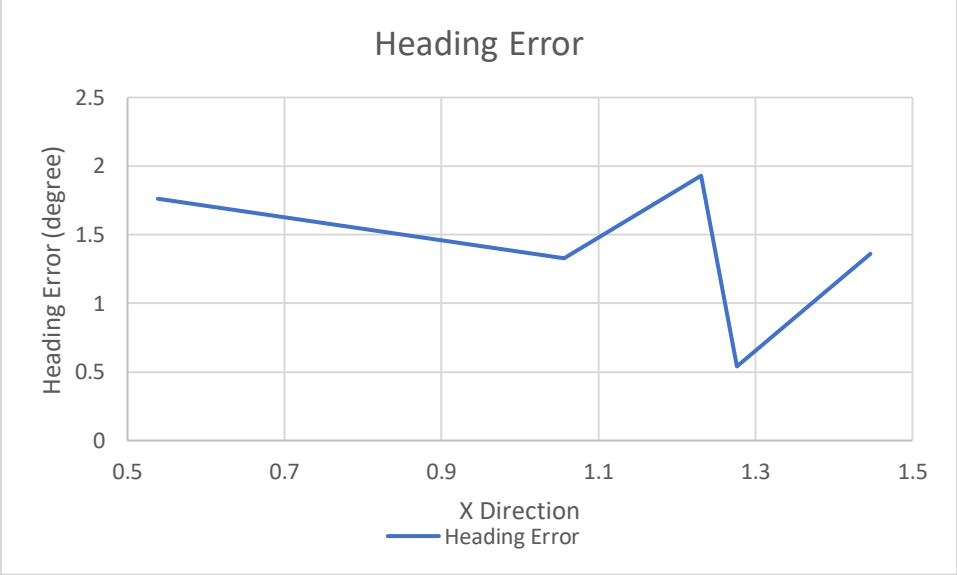


Figure 13. Heading phenomenon

These phenomena might generate large errors, and significantly reduce the navigability of legged robots. Therefore, a closed-loop method will be used, to determine whether the errors can be reduced.

2.2 Methodology

The coordinates for the hexapod are shown in Fig. 14.

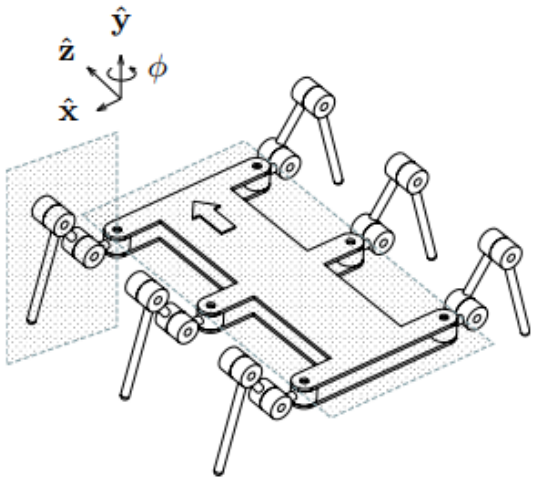


Figure 14. The coordinates of the hexapod.

This method uses the closed-loop control, due to the error capability of open loop control in practice. When the robot is rotated, there are always actual, $\Delta x_a, \Delta z_a, \Delta \Phi_a$, and desired displacement, $\Delta x_d, \Delta z_d, \Delta \phi_d$, where $\Delta x, \Delta y, and\ \Delta z$ are respected to x, y, z-directions, and $\Delta \phi$ is the heading desired [26]. The error correction is measured by equations below:

$$e_x = X_{i-1} - X_i - \Delta x_d \quad (1)$$

$$e_z = Z_{i-1} - Z_i - \Delta z_d \quad (2)$$

$$e_\phi = \phi_{i-1} - \phi_i - \Delta \phi_d \quad (3)$$

The heading and drifting can be corrected and prevented from propagating by adjusting $\Delta x_{d_{i+1}}, \Delta z_{d_{i+1}}, \Delta \phi_{d_{i+1}}$ by computing $e_{x_i}, e_{z_i}\ and\ e_{\phi_i}$. The planning displacement can be computed as below.

$$\Delta x_{i+1} = \left(1 + \frac{e_x}{\Delta x_{d_i}}\right) \Delta x_{d_{i+1}} \tag{4}$$

$$\Delta z_{i+1} = \left(1 + \frac{e_z}{\Delta z_{d_i}}\right) \Delta z_{d_{i+1}} \tag{5}$$

$$\Delta \phi_{i+1} = \left(1 + \frac{e_\Phi}{\Delta \Phi_{d_i}}\right) \Delta \Phi_{d_{i+1}} \tag{6}$$

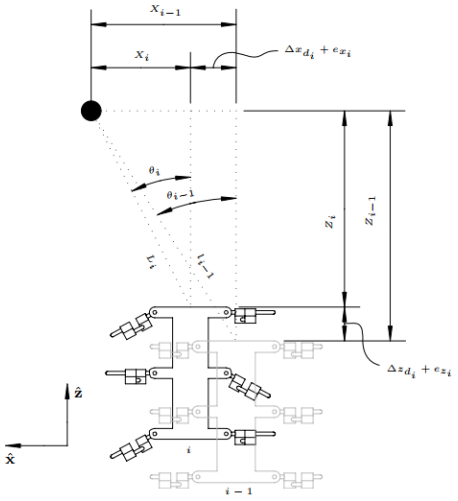


Figure 15. Illustration of the drift of the hexapod.

The eq. (4), (5) and (6) show that the next desired displacement will be corrected by the ratio of the current correction and the current desired displacement. Those equations help the next desired displacement getting close to the upcoming actual displacement. The closed-loop control block diagram can be illustrated as in Figure 16.

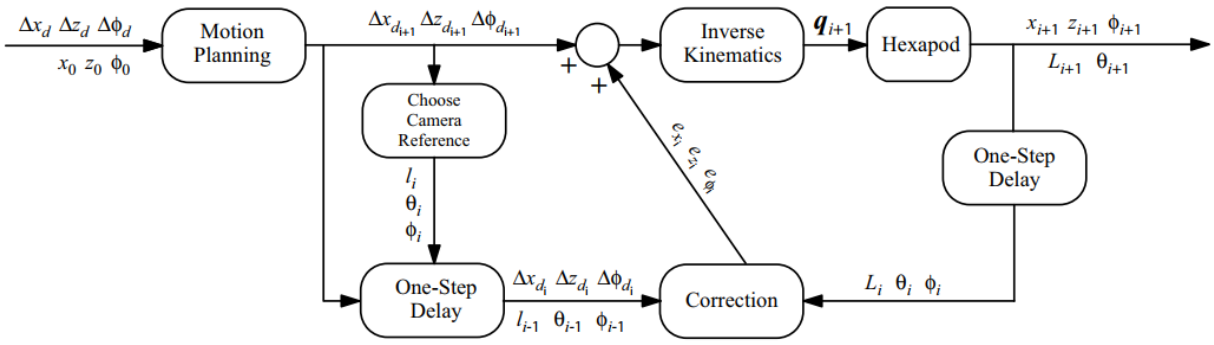


Figure 16. Block diagram.

2.3 Method Application and Algorithm

In this experiment, there are two experiments performed to verify the method: heading and drifting tests. By substituting eq. (1), (2), and (3) into (4), (5), and (6), the corrections can be simplified as:

$$\Delta x_{i+1} = (x_{i-1} - x_i)\Delta x_{d_{i+1}}/\Delta x_{d_i} \tag{7}$$

$$\Delta z_{i+1} = (z_{i-1} - z_i)\Delta z_{d_{i+1}}/\Delta z_{d_i} \tag{8}$$

$$\Delta \phi_{i+1} = (\phi_{i-1} - \phi_i)\Delta \Phi_{d_{i+1}}/\Delta \Phi_{d_i} \tag{9}$$

The leg design of the hexapod used only has 2 DOF, while the one used, in [26], has 3 DOF. Therefore, the current design has less capability of horizontal locomotion, which is x direction in this case. The estimation of the total displacement, Δd, can be used to replace the two correction of linear displacements, in eq. (7) and (8), and the camera reference is needed. Fortunately, Δd can be easily measured by the distance sensor. The correction for Δd can be computed as:

$$\Delta d_{i+1} = (d_{i-1} - d_i)\Delta d_{d_{i+1}}/\Delta d_{d_i} \tag{10}$$

Where d is the distance between hexapod and the object. In this experiment, the camera is used to detect and measure the difference between the center of the hexapod and the edge of target. The difference is measured by getting the difference in pixels between two centers. To convert pixel into angle, the ratio of pixel and angle can be used.

$$\tan\theta = \tan(\theta_{max}/2) * \frac{p}{p_{max}/2} \tag{11}$$

Where θ_{max} is the maximum angle of the camera that can capture, p_{max} is the maximum pixels that camera can output, and p is the pixel difference between the center of hexapod and the edge of object. The previous data will be stored for next computations.

For the simple control method, a certain angle is directly applied for every legs of active pair for operation, and the corrections are not explicitly calculated for each leg. However, if the corrections are simultaneously applied for every legs of active pair, the drift error will not be fixed. Thus, the correction might be reasonably applied on only one side of the active pair. The side of the drift is detected by referring the object to the camera reference, the center of the camera. Still, the problem on the corrections is that there are multiple difference-in-planar corrections that are normally computed. In the approach test, there are two corrections. If both corrections are combined with the turning angle, the effect of the correction can be diminished, or the error of heading can be worsened. Thus, in this experiment, the only correction used is the angular correction, $\Delta\phi$. In the heading test, the problem is similar. However, the $\Delta\phi_i$ can be used as the turning angle because it directly affects to the turning angle, and the correction Δd_i can be directly used to fix the error.

Due to the change above, the block diagram has a little difference with the original one. Δd_i will replace for both Δx_i and Δz_i and e_d will replace for both e_x and e_z. The new block diagram is shown in Figure 17.

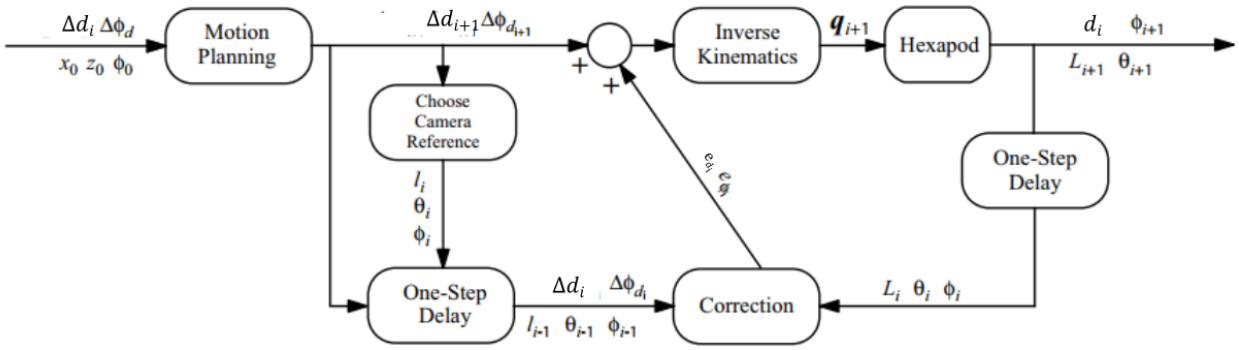


Figure 17. New Block Diagram

Chap 3

3.1 Experimental Procedures

The hexapod is placed at a certain point, which is around 120 cm away from the object and has around 25 degree shifted from the edge of the object. It will initially try to locate the object by turning, the turning phase. Then, it will approach to the object until it is around 30 cm away from it, the approaching phase. The hexapod also stops when the object is out of range of the ultrasonic sensor. During the experiment, the edge of the object is chosen as the camera reference point. In order to collect the results calculated by the hexapod, a laptop is directly connected to the hexapod. The results are displayed on a Serial display board, see Appendix II for more information. The data, then, are plotted in Excel with desired and actual values. The experiment is implemented in a closed room area, and the object chosen is a small blue box because it is easier for the camera to detect blue light in this room. The setup is shown in Figure 18. The experiment will be implemented for both controllers, open loop and closed loop, to have a comparison of the results. Additionally, to achieve statistical results, the experiment will be implemented 5 times for each controller.

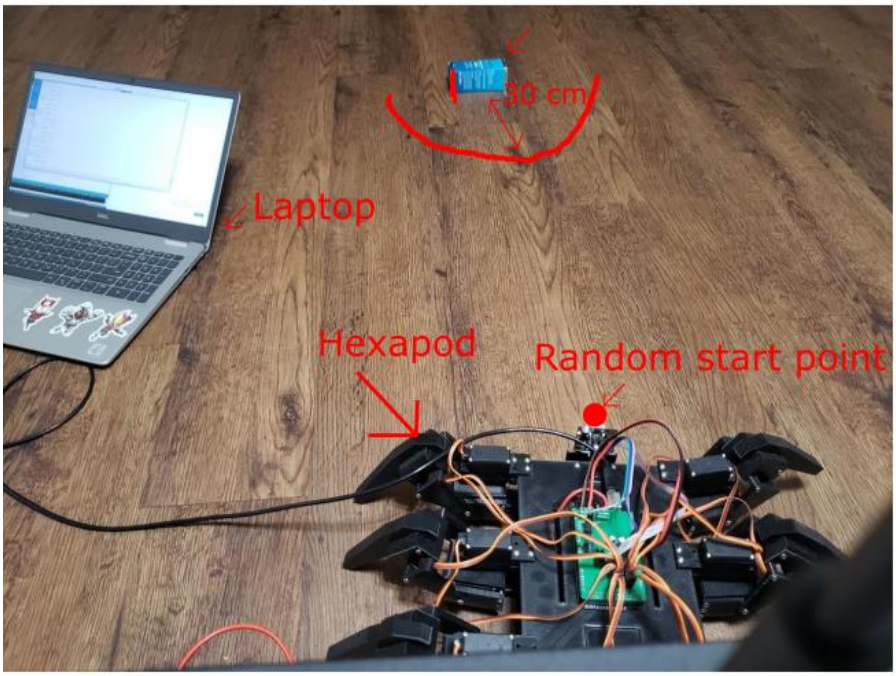


Figure 18. Experiment Setup.

3.2 Experimental Limitation

Due to the limitation of hardware, there are some limit intervals to be set for the controller, so that the hexapod can work smoothly and efficiently. The maximum turning angle interval of the legs is [-15,15], in degrees, the rising angle is fixed to 25 degree. The turning angle initially starts at 5 degree. Additionally, in the controller algorithm, the ultrasonic sensor data will be filtered by a condition, that current distance cannot be 5 cm more displacement difference than the previous distance. This is to eliminate the incorrect data received by the ultrasonic sensor in the room area.

See Appendix III for the Arduino code used in this experiment.

Chap 4

4. Results and Discussions

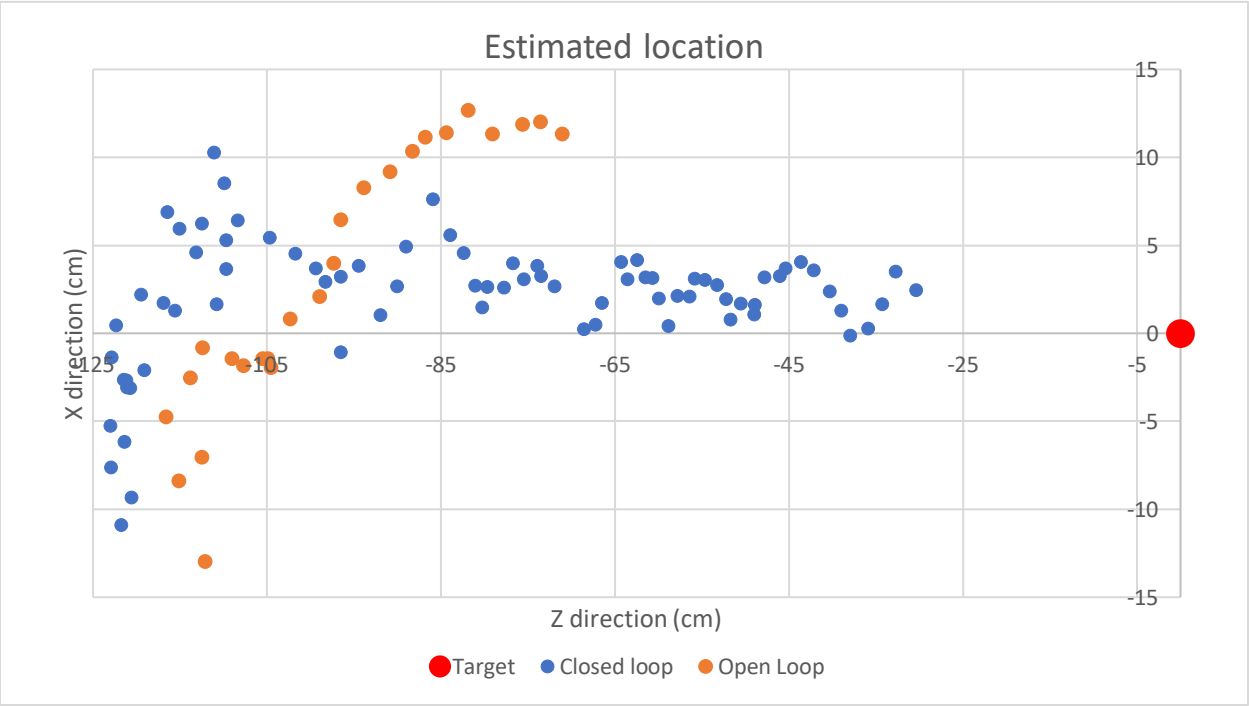


Figure 19. Estimated location of the hexapod respected to the object.

Figure 19 shows that, with the closed loop system, the hexapod was able to keep track of the object until it reached around 30 cm away, while the hexapod in the experiment of open loop system could not keep track of the object after a short approaching distance. The hexapod has a significant improvement on tracking the object by alleviating the errors.

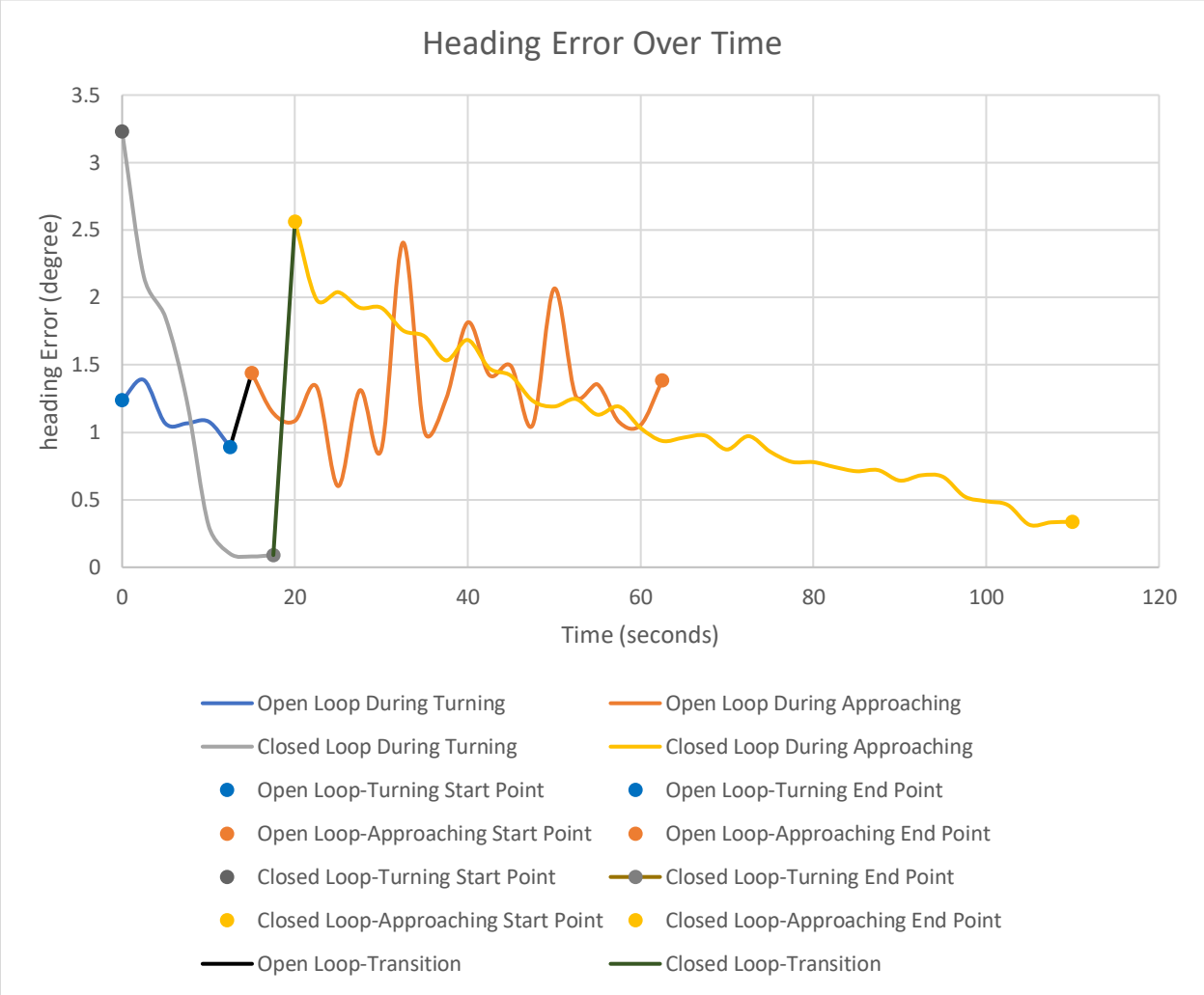


Figure 20. Average heading error over time

Figure 20 shows the heading error of $\Delta\Phi$ and $\Delta\Phi_{desired}$. The average heading errors of the open loop system are likely fluctuating around a certain error. In contrast, the average heading error of closed loop system gradually decrease over time. The absolute average heading error for the closed loop and open loop controllers are 0.88 degree and 1.3 degree, respectively. The closed loop system improves around 32% from the open loop system. Notice that there is gap between turning and approaching phases in Figure 20. It represents the transition from the

turning to the approaching phase. The collected data from the turning phase is reset when the approaching phase begins.

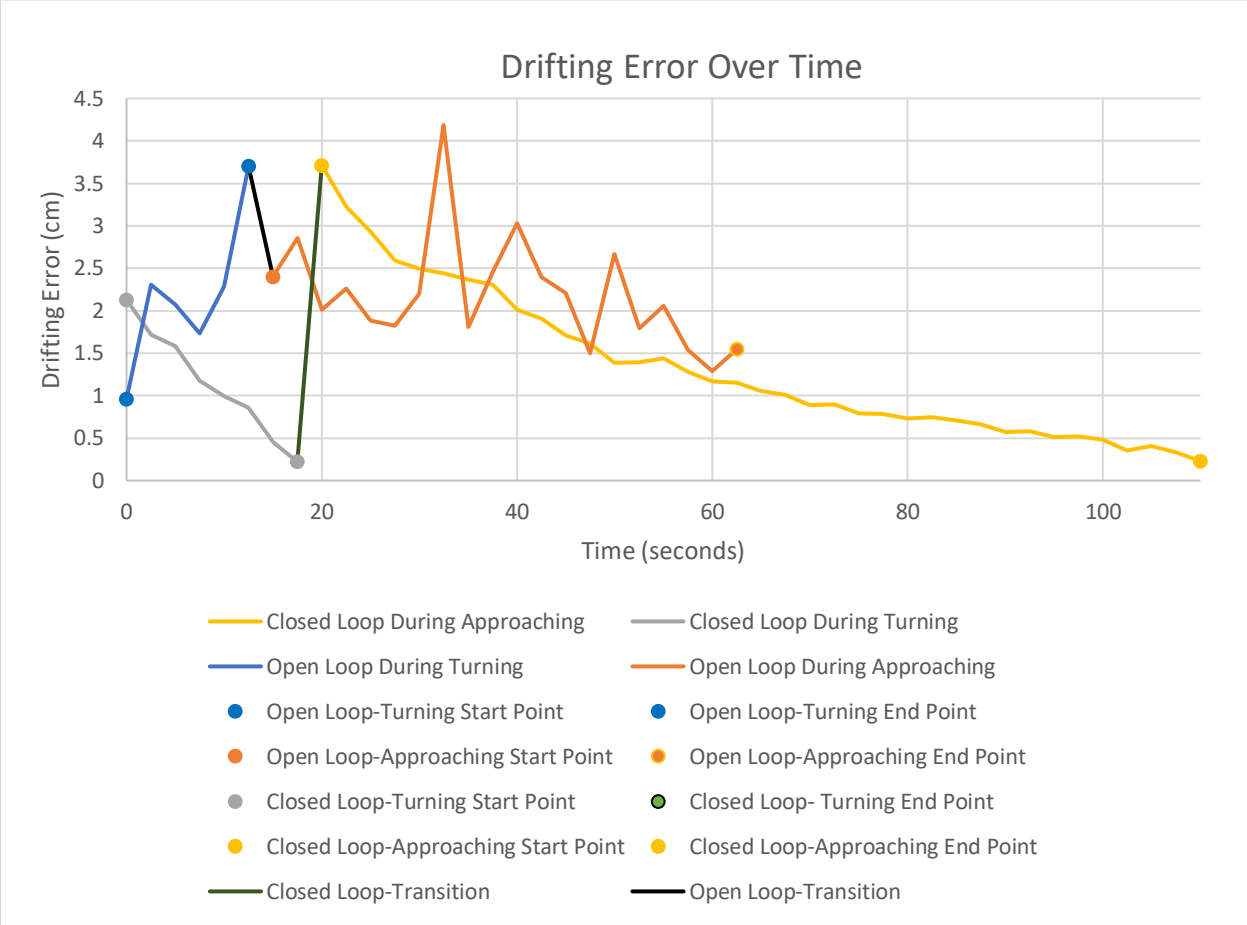


Figure 21. Average drifting error over time

Figure 21 shows the drifting error of Δd and $\Delta d_{desired}$. The results of drifting errors are similar to the results of heading errors. The improvement in closed loop control can be easily observed. The hexapod likely alleviated the overall drifting error for each footstep. The average drifting errors of the open loop control, in Figure 21, are likely fluctuating around a certain error, while the average drifting errors of the closed loop control and gradually decrease along with more steps taken. The absolute average drifting error of the closed loop system is only around

1.24 cm, while the absolute average drifting error of the open loop system is around 2.15 cm. It improves around 43%.

Chap 5

5.1 Conclusions

The primary purpose of this paper is to test the effectiveness of the proposed method in alleviating the drifting and heading error caused by imprecise hardware and a simple control method. By using a hexapod controlled by servos, camera, and an ultrasonic sensor, the result shows that the method works well in alleviating the drifting and heading errors, showing significant improvement in the navigability of an imprecise hexapod with a simple control. The method is modified somewhat from the original approach to work better with the chosen hardware. Although small errors still exist, the proposed approach is effective in controlling the locomotion of the hexapod.

5.2 Future work

For the future work, more improvements can be made to improve the controller performance and navigability. The hardware can be improved and updated. Instead of using servo motors, DC motors can be used to control the hexapod. A gearbox can be added to produce smaller angle rotations and better control. More sensors can be added to precisely measure the environment, such as encoder, force sensor. The design needs to be updated along with these changes and be able to move on different terrains. In additional, springs can also be used to reduce the impact while moving.

Appendix I

This section highlights the mechanism of the tripod gait. Tripod gait will split six legs of the hexapod into two groups, each group has three legs moving simultaneously. To facilitate the movement, it can be separated into five small steps: leveling up, leveling down, moving upward, moving backward, and being idle, respectively. A rotation of moving forward movement can be described that the first pair of legs rises upward, the second pair of legs moves backward subsequently. Then, the first pair moves forward, while the second pair is being idle. The first pair levels downward, and the second pair rises upward. The first pair moves backward, and the second pair moves forward. Eventually, the first pair is being idle, while the second pair levels downward. The new rotation will start the first pair with the first step, leveling up, and the second pair with the last step, being idle. Noticing that, the first step of one pair will be followed up with the fourth step of the other pair. However, at the initial state, all of six legs are touching the ground, that means the fourth step cannot be executed mechanically. To solve that issue, the fourth step of the second pair should be replaced with the last step of the movement, initially. The Figures 22 and 23 below will help illustrating the rotation of movement. There are four different rotations of movement can be created: walking forward, walking backward, turning left, turning right. However, in this experiment, walking forward and turning left are needed. For turning left, legs on the left side are moving backward, while legs on the right side are moving forward. Figures 24 and 25 illustrate how turning left works with tripod gait.

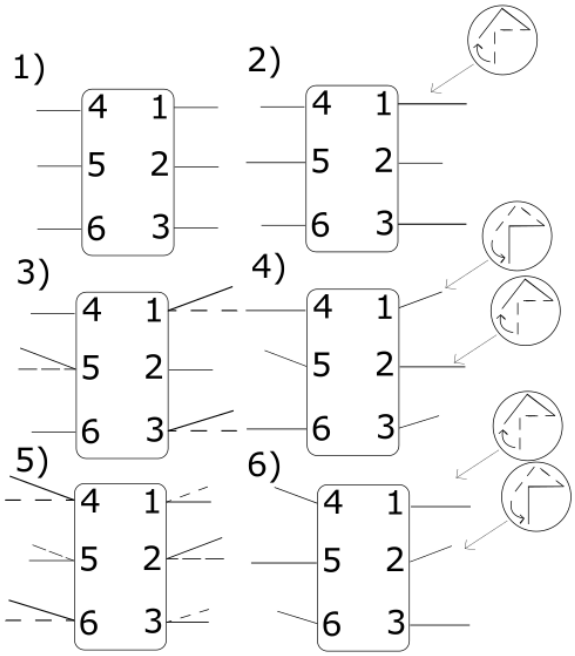


Figure 22. Move Forward

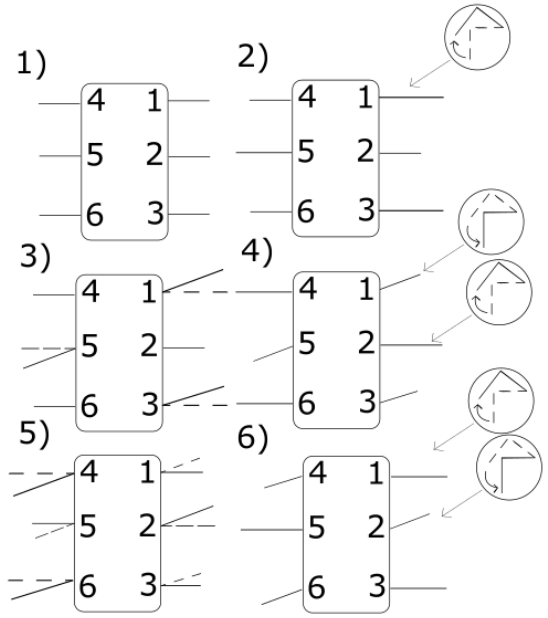


Figure 23. Turn Left

The idle pair and moving pair are respectively called inactive and active pair. Appendix I shows how that algorithm for movement works.

Appendix II

This section shows software used during the experiment, and current and data flow for building the hardware. Arduino IDE and PixyMon v2 are used for the microcontroller and camera, respectively. Arduino IDE is used to upload code to the microcontroller and receive the data from the camera. PixyMon v2 is used to display what camera is observing. PixyMon v2 will help figuring out what is going during testing. However, it is not necessarily needed for the experiment. The Serial board will show crucial data for the result plotting.

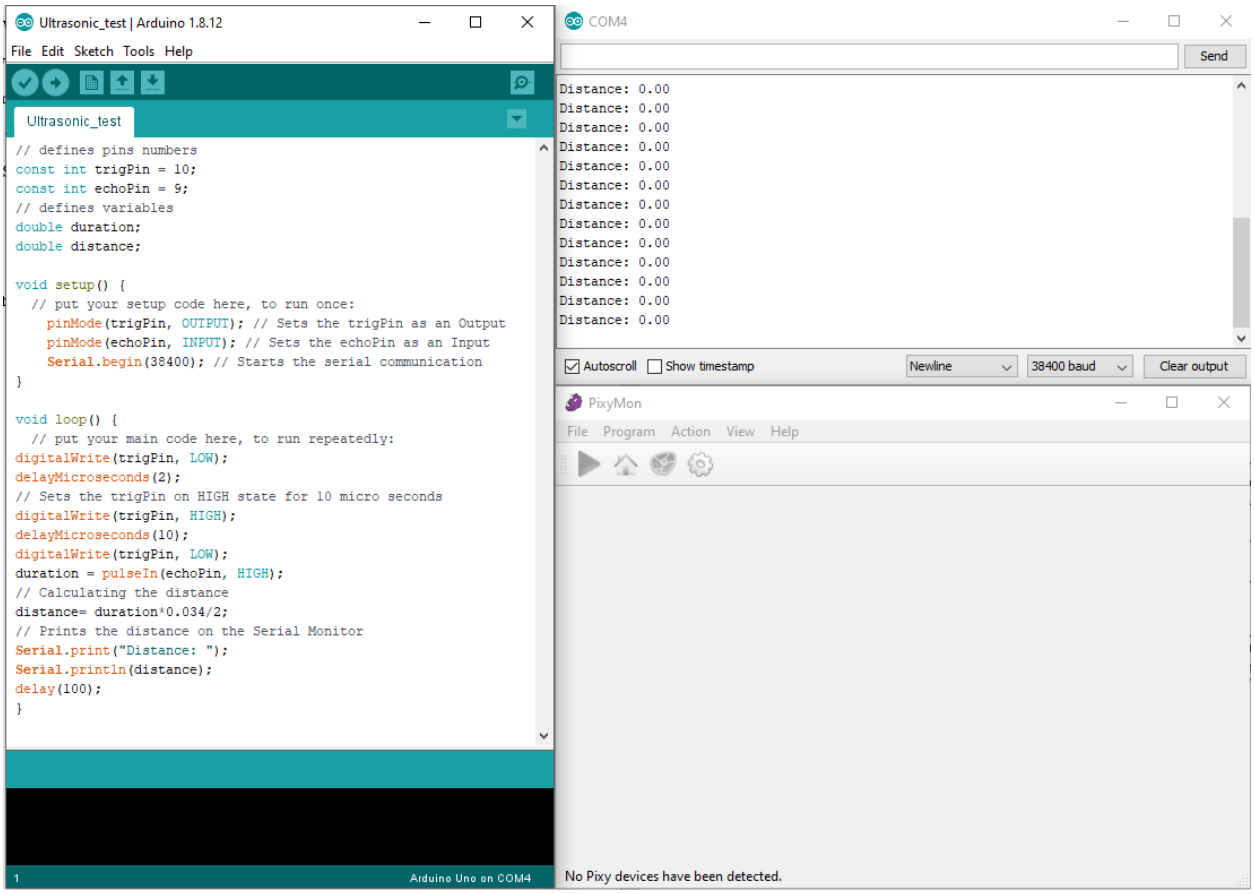
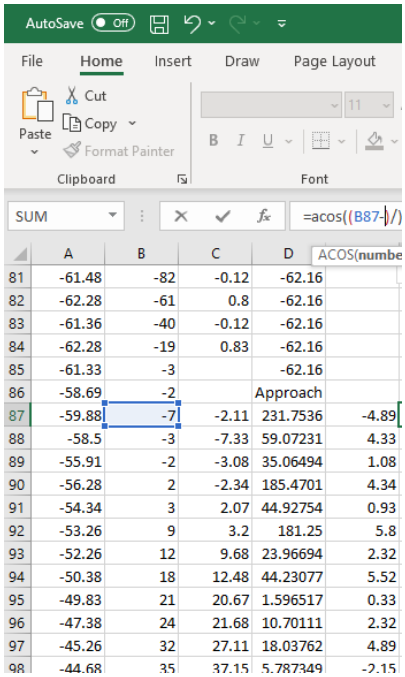


Figure 24. Arduino IDE and PixyMon v2.

The result will be recorded and plotted in Excel. Some calculations are performed directly in Excel, to calculate expected values, shown in Figure 27.



	A	B	C	D	
81	-61.48	-82	-0.12	-62.16	
82	-62.28	-61	0.8	-62.16	
83	-61.36	-40	-0.12	-62.16	
84	-62.28	-19	0.83	-62.16	
85	-61.33	-3		-62.16	
86	-58.69	-2		Approach	
87	-59.88	-7	-2.11	231.7536	-4.89
88	-58.5	-3	-7.33	59.07231	4.33
89	-55.91	-2	-3.08	35.06494	1.08
90	-56.28	2	-2.34	185.4701	4.34
91	-54.34	3	2.07	44.92754	0.93
92	-53.26	9	3.2	181.25	5.8
93	-52.26	12	9.68	23.96694	2.32
94	-50.38	18	12.48	44.23077	5.52
95	-49.83	21	20.67	1.596517	0.33
96	-47.38	24	21.68	10.70111	2.32
97	-45.26	32	27.11	18.03762	4.89
98	-44.68	35	37.15	5.787349	-2.15

Figure 25. Excel computing

The camera will detect the presence of the object and send the coordinates of the object in pixels. The ultrasonic sensor will measure the distance between the hexapod and object. The microcontroller will receive the data, calculate for adjustment angles, and send it to servos. The data and current flow are shown in Figure 28 and 29.

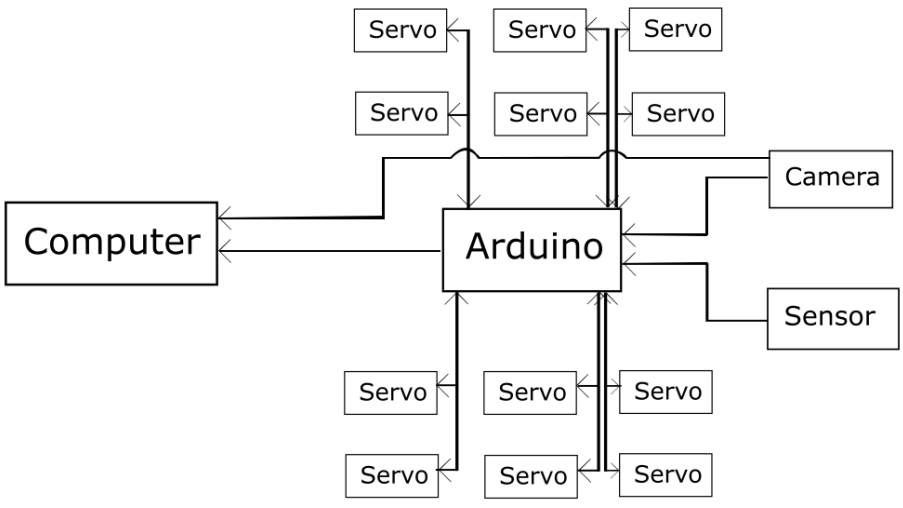


Figure 26. Data flow.

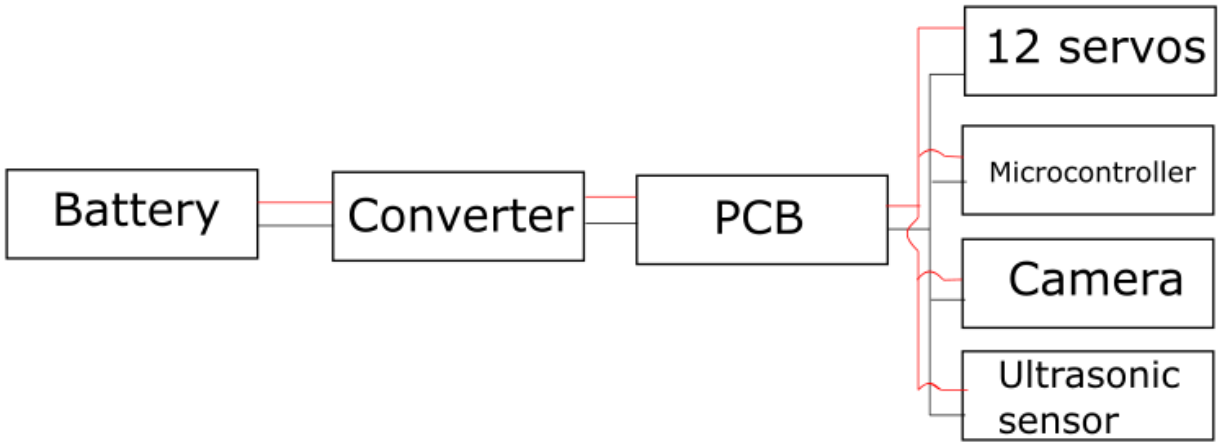


Figure 27. Power flow.

Appendix III

Main file:

```
#include <Arduino.h>
#include <Servo.h>
#include <SoftwareSerial.h>
#include <Pixy2.h>
#include "HexaLeg.hpp"
#include "HexaPair.hpp"

Servo LS11;               // LS11 = Left_Servo_1st Leg_1st Servo
Servo LS21;
Servo LS31;
Servo LS12;
Servo LS22;
Servo LS32;
Servo RS11;
Servo RS21;
Servo RS31;
Servo RS12;
Servo RS22;
Servo RS32;

Pixy2 pixy;
//For Operation
int stepth = 1;
double changingAngle[6] = {0,0,0,0,0,0};
int addUp1 = 0;
int addUp2 = 0;
boolean driftLeft = LOW;
boolean driftRight = LOW;
boolean inRange = HIGH;
double prevchangingAngle = 0;
boolean changState = LOW;

//For Camera
const int trig = 7;
const int echo = 6;
double duration = 0, distance = 0;
double dx[6] = {0,0,0,0,0,0}, dp[6] = {0,0,0,0,0,0};
double dz = 0;
double prevDegree[6] = {3,3,3,3,3,3};
int frame_dif;
int prevFrame = 0;
int x, W;
double init_distance = distance;
double d_distance; //desired distance
double prevDistance;
double prev_d_distance[6] = {0,0,0,0,0,0};
const double Length = 7.55;
const double maxLength = Length*sin(15*3.14/180);
const double minLength = Length*sin(3*3.14/180);
double dFrame = 0;

// For Movement
boolean PStatus = HIGH;                    // To stop pair 2 running at the 1st 2 steps
boolean RunStatus = HIGH;
boolean breakStatus = LOW;
const double riseAngle =  25.0;
int rotateAngle[6] = {0,0,0,0,0,0};
int Angle[6][2] = {{0,0},{0,0},{0,0},{0,0},{0,0},{0,0}};
int angleRatio[6]= {5,5,5,5,5,5};          // {2,2,3,3,3,3} Mim Ang;
int count_pair1 = 1;
int count_pair2 = 4;
int m = 1;
int m_prev = 3;
int init_step = 0;
int pside = 1;
int prevmovingCheck = 0;
boolean NonDetect = LOW;

// Initial Anngle for each servos
const int LS11Angle = 90;
```

```
const int LS12Angle = 100;
const int LS21Angle = 85;
const int LS22Angle = 87;
const int LS31Angle = 90;
const int LS32Angle = 110;
const int RS11Angle = 92;
const int RS12Angle = 90;
const int RS21Angle = 80;
const int RS22Angle = 110;
const int RS31Angle = 80;
const int RS32Angle = 95;

//DEFINE LEGS AND PAIRS
Hexaleg LegL1(&LS11,&LS12,Angle[0][0],Angle[0][1],
LS11Angle, LS12Angle,LOW);
Hexaleg LegL2(&LS21,&LS22,Angle[1][0],Angle[1][1],
LS21Angle, LS22Angle,LOW);
Hexaleg LegL3(&LS31,&LS32,Angle[2][0],Angle[2][1],
LS31Angle, LS32Angle,LOW);
Hexaleg LegR1(&RS11,&RS12,Angle[3][0],Angle[3][1],
RS11Angle, RS12Angle,HIGH);
Hexaleg LegR2(&RS21,&RS22,Angle[4][0],Angle[4][1],
RS21Angle, RS22Angle,HIGH);
Hexaleg LegR3(&RS31,&RS32,Angle[5][0],Angle[5][1],
RS31Angle, RS32Angle,HIGH);

Hexapair fPair(LegL1,LegR2,LegL3);
Hexapair sPair(LegR1,LegL2,LegR3);
void setup() {
 // put your setup code here, to run once:
 Serial.begin(38400);
 delay(20);
 //L Leg 1
 LS11.attach(39,900,2100); // 900 = 0 degree; 2100 = 180 degrees
 LS12.attach(38,900,2100);
 //L Leg 2
 LS21.attach(37,900,2100);
 LS22.attach(36,900,2100);
 //L Leg 3
 LS31.attach(35,900,2100);
 LS32.attach(34,900,2100);
 //R Leg 1
 RS11.attach(28,900,2100);
 RS12.attach(29,900,2100);
 //R Leg 2
 RS21.attach(30,900,2100);
 RS22.attach(31,900,2100);
 //R Leg 3
 RS31.attach(32,900,2100);
 RS32.attach(33,900,2100);

 //Initial Position
 fPair.initialState();
 sPair.initialState();

 //Init Camera and Sensor
 //pinMode(trig, OUTPUT);
 //pinMode(echo, INPUT);
 pixy.init();
 pixy.setLamp(HIGH,HIGH);
 delay(1000);
}
void loop() {
 //Camera
 //Serial.print("Start detecting Object \n");
 operation();
 //Movement
 checkPos();
 if(changState == HIGH) {operation();changState = LOW;}
 updateAngle();
```

```
if(stepth == 1)
 {
 Serial.print(" ------------------------------------------------\n");
  Serial.print(" rotateAngle[1] = ");
  Serial.print(rotateAngle[1]);
  Serial.print("\t");
  Serial.print(" rotateAngle[4] = ");
  Serial.print(rotateAngle[4]);
  Serial.print("\t");
  Serial.print(" changingAngle[4] = ");
  Serial.print(prevchangingAngle);
  Serial.print("\n");
  Serial.print(" prev_d_distance[0] = ");
  Serial.print(prev_d_distance[0]);
  Serial.print("\t");
  Serial.print(" prev_d_distance[3] = ");
  Serial.print(prev_d_distance[3]);
  Serial.print("\n");
  Serial.print(" prevDistance = ");
  Serial.print(prevDistance);
  Serial.print("\n");
 Serial.print(" ------------------------------------------------\n");
 }
 else
 {
 Serial.print(" ------------------------------------------------\n");
  Serial.print(" Angle[0][0] = ");
  Serial.print(Angle[0][0]);
  Serial.print("\t");
  Serial.print(" Angle[4][0] = ");
  Serial.print(Angle[4][0]);
  Serial.print("\t");
  if (driftLeft == HIGH)
  {
  Serial.print(" changingAngle[1] = ");
Serial.print(prevchangingAngle);
   Serial.print("\n");
   }
   else
   {
   Serial.print(" changingAngle[4] = ");
   Serial.print(prevchangingAngle);
   Serial.print("\n");
   }
 Serial.print(" ------------------------------------------------\n");
 }
 while(RunStatus){
  //Serial.print("in RunStatus\n");
  updateAngle();
  hexapodMovement(m);
 }
 while(true)
 {
  if(distance <30)
  {
    reverse_step();
    delay(1000);
  }
  else break;
 }

}
void updateAngle(){
   //Serial.print("in updateAngle\n");
 Angle[0][1] = riseAngle;
Angle[0][0] = rotateAngle[0] + changingAngle[0];
 Angle[1][1] = riseAngle;
 Angle[1][0] = rotateAngle[1] + changingAngle[1];
 Angle[2][1] = riseAngle;
 Angle[2][0] = rotateAngle[2] + changingAngle[2];
```

```
Angle[2][1] = riseAngle;
  Angle[2][0] = rotateAngle[2] + changingAngle[2];
  Angle[3][1] = riseAngle;
  Angle[3][0] = rotateAngle[3] + changingAngle[3];
  Angle[4][1] = riseAngle;
  Angle[4][0] = rotateAngle[4] + changingAngle[4];
  Angle[5][1] = riseAngle;
  Angle[5][0] = rotateAngle[5] + changingAngle[5];

  prevchangingAngle = changingAngle[4];
  if(driftLeft == HIGH) prevchangingAngle = changingAngle[1];

       changingAngle[0] = 0;
       changingAngle[1] = 0;
       changingAngle[2] = 0;
       changingAngle[3] = 0;
       changingAngle[4] = 0;
       changingAngle[5] = 0;
  if (Angle[0][0] >15)
  {
    Angle[0][0] = 15;
    }
if (Angle[1][0] >15)
  {
    Angle[1][0] = 15;
    }
  if (Angle[2][0] >15)
  {
    Angle[2][0] = 15;
    }
  if (Angle[3][0] >15)
  {
    Angle[3][0] = 15;
    }
  if (Angle[4][0] >15)
{
    Angle[4][0] = 15;
    }
  if (Angle[5][0] >15)
  {
    Angle[5][0] = 15;
    }

  if (Angle[0][0] <0)
  {
    Angle[0][0] = 0;
    }
  if (Angle[1][0] <0)
  {
    Angle[1][0] = 0;
    }
  if (Angle[2][0] <0)
  {
    Angle[2][0] = 0;
    }
  if (Angle[3][0] <0)
  {
    Angle[3][0] = 0;
    }
  if (Angle[4][0] <0)
  {
    Angle[4][0] = 0;
    }
  if (Angle[5][0] <0)
  {
    Angle[5][0] = 0;
    }
  prevDegree[0] = Angle[0][0];
  prevDegree[1] = Angle[1][0];
  prevDegree[2] = Angle[2][0];
```

```
prevDegree[3] = Angle[3][0];
  prevDegree[4] = Angle[4][0];
  prevDegree[5] = Angle[5][0];

  prev_d_distance[0] = sin(Angle[0][0]*3.14/180)*Length;
  prev_d_distance[1] = sin(Angle[1][0]*3.14/180)*Length;
  prev_d_distance[2] = sin(Angle[2][0]*3.14/180)*Length;
  prev_d_distance[3] = sin(Angle[3][0]*3.14/180)*Length;
  prev_d_distance[4] = sin(Angle[4][0]*3.14/180)*Length;
  prev_d_distance[5] = sin(Angle[5][0]*3.14/180)*Length;

}

void mcheck(){

  fPair.pcheck(count_pair1);
  sPair.pcheck(count_pair2);
  if(count_pair1 == 3 && init_step == 0 && fPair.pStatus ==
HIGH){      //Start second pairs after initial step
    PStatus = LOW;
    init_step++;
    }

  if (fPair.pStatus == HIGH)
  {
    //Serial.print("F Pair Status \t HIGH\n");
    sPair.turnOff();
    count_pair1++;
    pside++;
    fPair.reset();
    breakStatus = HIGH;
    }
  if (sPair.pStatus == HIGH)
  {
   // Serial.print("S Pair Status \t HIGH\n");
fPair.turnOff();
    count_pair2++;
    pside++;
    sPair.reset();
    breakStatus = HIGH;
    }
  if(pside > 2){
    pside = 1;
    }
  if (count_pair1 > 5){
    count_pair1 = 1;                    // Reset count
    }
  if (count_pair2 > 5){
    count_pair2 = 1;                    // Reset count
  }
  if (init_step > 2){
    init_step = 1;
    }
  if ( ((count_pair1 == 5 && count_pair2 == 3) || (count_pair2 ==
5&& count_pair1 == 3)) && init_step != 0 && breakStatus ==
HIGH){
    RunStatus = LOW;
    }
  }
void Forward(){
      //Serial.print("in Forward \n");
    mcheck();
    if (fPair.pStatus == LOW && pside%2 == 1){
      LegL1.moveForward(count_pair1);
      LegR2.moveForward(count_pair1);
      LegL3.moveForward(count_pair1);
    }
   if(sPair.pStatus == LOW && pside%2 == 0){
    if (PStatus == LOW){
      LegR1.moveForward(count_pair2);
```

```
LegL2.moveForward(count_pair2);
      LegR3.moveForward(count_pair2);
     }
     else{
       sPair.turnOn();
       fPair.turnOff();
       }
   }
}
void Backward(){
     //Serial.print("in Backward \n");
  mcheck();
   if (fPair.pStatus == LOW && pside%2 == 1){
     LegL1.moveBackward(count_pair1);
     LegR2.moveBackward(count_pair1);
     LegL3.moveBackward(count_pair1);
   }
  if(sPair.pStatus == LOW && pside%2 == 0){
   if (PStatus == LOW){
     LegR1.moveBackward(count_pair2);
     LegL2.moveBackward(count_pair2);
     LegR3.moveBackward(count_pair2);
     }
     else{
       sPair.turnOn();
       fPair.turnOff();
       }
   }
}
void ToLeft(){
mcheck();
   if (fPair.pStatus == LOW && pside%2 == 1){
     LegL1.moveBackward(count_pair1);
     LegR2.moveForward(count_pair1);
     LegL3.moveBackward(count_pair1);
}
  if(sPair.pStatus == LOW && pside%2 == 0){
   if (PStatus == LOW){
     LegR1.moveForward(count_pair2);
     LegL2.moveBackward(count_pair2);
     LegR3.moveForward(count_pair2);
     }
     else{
       sPair.turnOn();
       fPair.turnOff();
       }
   }
}
void ToRight(){
     //Serial.print("in ToRight \n");
  mcheck();
   if (fPair.pStatus == LOW && pside%2 == 1){
     LegL1.moveForward(count_pair1);
     LegR2.moveBackward(count_pair1);
     LegL3.moveForward(count_pair1);
   }
  if(sPair.pStatus == LOW && pside%2 == 0){
   if (PStatus == LOW){
     LegR1.moveBackward(count_pair2);
     LegL2.moveForward(count_pair2);
     LegR3.moveBackward(count_pair2);
     }
     else{
       sPair.turnOn();
       fPair.turnOff();
       }
   }
}
void hexapodMovement (int m) {
  switch (m) {
```

```
//Move Forward
  case 2:{
    Forward();
      break;
  }
  //Move To Left
  case 3:{
    ToLeft();
      break;
  }
  //Move To Right
  case 4:{
    ToRight();
      break;
  }
  //Move Backward
  case 5:{
    Backward();
      break;
  }
  case 6:{
    ToLeft();
      break;
  }
  default:{
    Serial.print("Error hexapodMovement \n");
      break;
    }
  }
}



int getFrameDif(int width,int m_x){
  int frame_dif = m_x - width/2;
  return frame_dif;
}
void calculating_Angle(boolean rotate, boolean changing){
  if (rotate == HIGH){
    if (init_step == 0)        //(prevFrame <= 0 || init_step == 0)
      {
        rotateAngle[0] = prevDegree[0];
        rotateAngle[1] = prevDegree[1];
        rotateAngle[2] = prevDegree[2];
        rotateAngle[3] = prevDegree[3];
        rotateAngle[4] = prevDegree[4];
        rotateAngle[5] = prevDegree[5];
        }
      else
      {
        dp[0] = prevDegree[0]/abs(prevFrame-
frame_dif)*abs(frame_dif);
        dp[1] = prevDegree[1]/abs(prevFrame-
frame_dif)*abs(frame_dif);
        dp[2] = prevDegree[2]/abs(prevFrame-
frame_dif)*abs(frame_dif);
        dp[3] = prevDegree[3]/abs(prevFrame-
frame_dif)*abs(frame_dif);
        dp[4] = prevDegree[4]/abs(prevFrame-
frame_dif)*abs(frame_dif);
        dp[5] = prevDegree[5]/abs(prevFrame-
frame_dif)*abs(frame_dif);
        Serial.print(" prevDegree[1] = ");
        Serial.print(prevDegree[1]);
        Serial.print("\n");
        Serial.print(" dp[1] = ");
        Serial.print(dp[1]);
        Serial.print("\n");
        Serial.print(" dp[3] = ");
        Serial.print(dp[3]);
        Serial.print("\n");
```

```
dx[0] = (prevDistance-distance)*abs(init_distance-
distance)/prev_d_distance[0];
      dx[1] = (prevDistance-distance)*abs(init_distance-
distance)/prev_d_distance[1];
      dx[2] = (prevDistance-distance)*abs(init_distance-
distance)/prev_d_distance[2];
      dx[3] = (prevDistance-distance)*abs(init_distance-
distance)/prev_d_distance[3];
      dx[4] = (prevDistance-distance)*abs(init_distance-
distance)/prev_d_distance[4];
      dx[5] = (prevDistance-distance)*abs(init_distance-
distance)/prev_d_distance[5];
      Serial.print(" dx[3] = ");
      Serial.print(dx[3]);
      Serial.print("\n");
      Serial.print(" dx[3] = ");
      Serial.print(asin(dx[3]/Length));
      Serial.print("\n");
      rotateAngle[0] = dp[0];
      rotateAngle[1] = dp[1];
      rotateAngle[2] = dp[2];
      rotateAngle[3] = dp[3];
      rotateAngle[4] = dp[4];
      rotateAngle[5] = dp[5];
      if(m == 3)
      {
       if(count_pair1 == 5 && count_pair2 == 3)
       {
        changingAngle[3] = asin(dx[3]/Length)*180/3.14;
        changingAngle[5] = asin(dx[5]/Length)*180/3.14;
       }
       if (count_pair1 == 3 && count_pair2 == 5)
       {
        changingAngle[4] = asin(dx[4]/Length)*180/3.14;
       }
}
      if(m == 4)
      {
       if(count_pair1 == 5 && count_pair2 == 3)
       {
        changingAngle[0] = asin(dx[0]/Length)*180/3.14;
        changingAngle[2] = asin(dx[2]/Length)*180/3.14;
       }
       if (count_pair1 == 3 && count_pair2 == 5)
       {
        changingAngle[1] = asin(dx[1]/Length)*180/3.14;
       }
      }
    }
  }

 if (changing == HIGH){
  if((frame_dif-dFrame) > 0){
   driftLeft = HIGH;
   driftRight = LOW;
   }
  else if ((frame_dif-dFrame) < 0){
   driftLeft = LOW;
   driftRight = HIGH;
   }
  else
   {
    driftLeft = LOW;
    driftRight = LOW;
   }
  if (init_step == 0)       //(prevFrame <= 0 || init_step == 0)
  {
    rotateAngle[0] = 15;
    rotateAngle[1] = 15;
    rotateAngle[2] = 15;
```

```
                    rotateAngle[3] = 15;
      rotateAngle[4] = 15;
      rotateAngle[5] = 15;
   }
   else
   {
    if (driftLeft == HIGH){
    Serial.print("driftLeft \n");
    Serial.print(" prevDegree[1] = ");
     Serial.print(prevDegree[1]);
     Serial.print("\n");
     dz = abs(pow(prevDistance,2)-pow(distance,2)-
pow(prev_d_distance[0],2))/(2*prev_d_distance[0]);
     dFrame = prevFrame*(dz+prev_d_distance[0])/dz;
    Serial.print(" dFrame = ");
     Serial.print(dFrame);
     Serial.print("\n");
     changingAngle[0] = prevDegree[0]/(dFrame-
frame_dif)*(frame_dif);
     changingAngle[1] = prevDegree[1]/(dFrame-
frame_dif)*(frame_dif);
     changingAngle[2] = prevDegree[2]/(dFrame-
frame_dif)*(frame_dif);
     changingAngle[3] = 0;
     changingAngle[4] = 0;
     changingAngle[5] = 0;
     }
    if (driftRight == HIGH){
    Serial.print("driftRight \n");
    Serial.print(" prevDegree[4] = ");
     Serial.print(prevDegree[4]);
     Serial.print("\n");
     dz = abs(pow(prevDistance,2)-pow(distance,2)-
pow(prev_d_distance[4],2))/(2*prev_d_distance[4]);
     dFrame = prevFrame*(dz+prev_d_distance[4])/dz;
Serial.print(" dFrame = ");
     Serial.print(dFrame);
     Serial.print("\n");
     changingAngle[0] = 0;
     changingAngle[1] = 0;
     changingAngle[2] = 0;
     changingAngle[3] = prevDegree[3]/(dFrame-
frame_dif)*(frame_dif);
     changingAngle[4] = prevDegree[4]/(dFrame-
frame_dif)*(frame_dif);
     changingAngle[5] = prevDegree[5]/(dFrame-
frame_dif)*(frame_dif);

     }
    if (driftRight == LOW && driftLeft == LOW){
     changingAngle[0] = 0;
     changingAngle[1] = 0;
     changingAngle[2] = 0;
     changingAngle[3] = 0;
     changingAngle[4] = 0;
     changingAngle[5] = 0;
     }
   }
  }
}

void detect_object(){
     Serial.print("\t In Detect_Object \n");
   // grab blocks!
   while(true)
   {
    pixy.ccc.getBlocks();
    if(pixy.ccc.numBlocks)
{
    Serial.print("Detected ");
```

```
Serial.println(pixy.ccc.numBlocks);
      Serial.print(" block ");
      Serial.print(0);
      Serial.print(": ");
      x = pixy.ccc.blocks[0].m_x;
      W = pixy.ccc.blocks[0].m_width;
      frame_dif = getFrameDif(pixy.frameWidth,x);
      Serial.print(" FrameDif = ");
      Serial.print(frame_dif);
      Serial.print("\n");
        break;
      }
    }
    // If there are detect blocks, print them!
    if ((count_pair1 == 5 && count_pair2 == 3) || (count_pair1 == 3
&& count_pair2 == 5) || init_step == 0)
    {
       NonDetect = LOW;
      while(true)
      {
        digitalWrite(trig,LOW);
        delayMicroseconds(2);
        digitalWrite(trig,HIGH);
        delayMicroseconds(2);
        digitalWrite(trig,LOW);
        duration = pulseIn(echo,HIGH);
        distance = duration/29/2;
        if (init_step == 0 || abs(prevDistance-distance) < 5) break;
      }
      delay(100);
      Serial.print(" Distance = ");
      Serial.print(distance);
      Serial.print("\n");
      if (init_step == 0)
{
        init_distance = distance;
      }
      calculating_Angle(HIGH,LOW);
        prevFrame = frame_dif;
        prevDistance = distance;
      if (frame_dif > 8)
        {

          m = 4;
        }
        else if (frame_dif < -8)
        {
          m = 3;
        }
        else{
          ++stepth;
          m = 2;
          return;
        }
    }
}
void approach_object(){
    Serial.print("\t In Approach_Object \n");
    while(true)
    {
      pixy.ccc.getBlocks();
  if(pixy.ccc.numBlocks)
      {
      Serial.print("Detected ");
      Serial.println(pixy.ccc.numBlocks);
      Serial.print(" block ");
      Serial.print(0);
      Serial.print(": ");
      x = pixy.ccc.blocks[0].m_x;
```

```
W = pixy.ccc.blocks[0].m_width;
      frame_dif = getFrameDif(pixy.frameWidth,x);
      Serial.print(" FrameDif = ");
      Serial.print(frame_dif);
      Serial.print("\n");
       break;
      }
    }
    // If there are detect blocks, print them!
    if ((count_pair1 == 5 && count_pair2 == 3) || (count_pair1 == 3
&& count_pair2 == 5) || init_step == 0)
    {
      NonDetect = LOW;
      while(true)
      {
       digitalWrite(trig,LOW);
       delayMicroseconds(2);
       digitalWrite(trig,HIGH);
       delayMicroseconds(10);
       digitalWrite(trig,LOW);
       duration = pulseIn(echo,HIGH);
       distance = duration/29/2;
       if (abs(prevDistance-distance) < 5) break;
      }
      delay(100);
      Serial.print(" Distance = ");
      Serial.print(distance);
      Serial.print("\n");
      if (init_step == 0)
      {
init_distance = distance;
      }
      calculating_Angle(LOW,HIGH);
        prevFrame = frame_dif;
        prevDistance = distance;;
}
  }
void operation(){
  //Serial.print("In Operation \n");
    switch (stepth){
      case 1:{
        //Serial.print("Case 1\n");
        detect_object();
        break;
        }
      case 2:{
        //Serial.print("Case 2\n");
        approach_object();
        break;
        }
      }
  }
void initState(){
  //Serial.print("In initState \n");
  if(stepth == 2){
  changingAngle[0] = 0;
  changingAngle[1] = 0;
  changingAngle[2] = 0;
  changingAngle[3] = 0;
  changingAngle[4] = 0;
  changingAngle[5] = 0;
  Angle[0][1] = 0;
  Angle[0][0] = 0;
  Angle[1][1] = 0;
  Angle[1][0] = 0;
  Angle[2][1] = 0;
  Angle[2][0] = 0;
  Angle[3][1] = 0;
  Angle[3][0] = 0;
  Angle[4][1] = 0;
```

```
  Angle[4][0] = 0;
  Angle[5][1] = 0;
  Angle[5][0] = 0;
  dx[0] = 0;
  dx[1] = 0;
  dx[2] = 0;
  dx[3] = 0;
  dx[4] = 0;
  dx[5] = 0;
  dp[0] =0 ;
  dp[1] =0 ;
  dp[2] =0 ;
  dp[3] =0 ;
  dp[4] =0 ;
  dp[5] =0 ;
  prevDegree[0] = 3;
  prevDegree[1] = 3;
  prevDegree[2] = 3;
  prevDegree[3] = 3;
  prevDegree[4] = 3;
  prevDegree[5] = 3;
  rotateAngle[0] = 0;
  rotateAngle[1] = 0;
  rotateAngle[2] = 0;
  rotateAngle[3] = 0;
  rotateAngle[4] = 0;
  rotateAngle[5] = 0;
   }
  // For Movement
  PStatus = HIGH;                    // To stop pair 2 running at
the 1st 2 steps
  count_pair1 = 1;
  count_pair2 = 4;
  init_step = 0;
  pside = 1;
  fPair.reset();
  sPair.reset();
}
void reverse_step(){
  Serial.print("In Reverse_step \n");
    if(count_pair1 == 5){
      fPair.turnOn();
      }
    else{
      if(count_pair1 == 2 || count_pair1 == 3){
        //Serial.print("Reverse Pair 1 \n ");
        sPair.initialState();
        fPair.initialState();
      }
      else{
        Serial.print("Something wrong \n ");
        Serial.print(" count_pair1 = ");
        Serial.print(count_pair1);
        Serial.print("\n");
        }
      }
    if(count_pair2 == 5){
      sPair.turnOn();
      }
    else{
      if(count_pair2 == 2 || count_pair2 == 3){
        //Serial.print("Reverse Pair 2 \n ");
        fPair.initialState();
        sPair.initialState();
      }
      else{
Serial.print("Something wrong \n ");
        Serial.print(" count_pair1 = ");
        Serial.print(count_pair1);
        Serial.print(" count_pair2 = ");
```

```
      Serial.print(count_pair2);
      Serial.print("\n");
      }
     }
}
void checkPos(){
 //Serial.print("In CheckPos \n");
 if (m_prev != m && init_step != 0){
  reverse_step();
  initState();
  changState = HIGH;
  Serial.print("Done reverse \n");
  }
 m_prev = m;
 RunStatus = HIGH;
 breakStatus = LOW;
 }
```

HexaLeg.cpp file:

```
#include <Servo.h>
#include <Arduino.h>
#include "Hexaleg.hpp"

void Hexaleg::finitState(){

 firstServo.write(finitialState);
 }
void Hexaleg::sinitState(){

 secondServo.write(sinitialState);
 }
void Hexaleg::initialState(){
 sinitState();
 finitState();
 }
void Hexaleg::reverse(){
    lowerLeg();
 }
void Hexaleg::reverse_check(){
 if (finitialState == firstServo.read()) hStatus = HIGH;
 }

void Hexaleg::turnOn(){
 hStatus = HIGH;
 }
int Hexaleg::readAng() {
 return elem22;
 }

void Hexaleg::riseLeg() {
 if (hStatus == HIGH) return;
```

```
  if (side == LOW){
    secondServo.write(secondServo.read()+elem21);
    delay(5);
  }
  if (side == HIGH){
    secondServo.write(secondServo.read()-elem21);
    delay(5);
  }
}
void Hexaleg::lowerLeg() {
  if (hStatus == HIGH) return;
  if (side == LOW){
    secondServo.write(secondServo.read()-elem21);
    delay(5);
  }
  if (side == HIGH){
    secondServo.write(secondServo.read()+elem21);
    delay(5);
  }
}
void Hexaleg::rotRight() {
  if (hStatus == HIGH) return;
  if (side == LOW){
    firstServo.write(firstServo.read()+elem21);
    delay(10);
  }
  if (side == HIGH){
    firstServo.write(firstServo.read()-elem21);
    delay(10);
  }
}
void Hexaleg::rotLeft() {
  if (hStatus == HIGH) return;
  if (side == LOW){
    firstServo.write(firstServo.read()-elem21);
    delay(5);
}
  if (side == HIGH){
    firstServo.write(firstServo.read()+elem21);
    delay(5);
  }
}
void Hexaleg::moveForward(int count_pair){
  angStream(count_pair);
  switch (count_pair){
    case 1:{
      riseLeg();
      break;
      }
    case 2:{
      rotRight();
      break;
      }
    case 3:{
      lowerLeg();
      break;
      }
    case 4:{
      rotLeft();
      break;
      }
    case 5:{
      turnOn();
      break;
    }
    default:{
      Serial.print("error in moveForward\n");
      break;
      }
    }
```

```
}

void Hexaleg::moveBackward(int count_pair){
 angStream(count_pair);
 switch (count_pair){
  case 1:{
   riseLeg();
   break;
   }
  case 2:{
   rotLeft();
   break;
   }
  case 3:{
   lowerLeg();
   break;
   }
  case 4:{
   rotRight();
   break;
   }
  case 5:{
   turnOn();
   break;
  }
  default:{
   Serial.print("error in moveBackward\n");
   break;
   }
  }
 }

void Hexaleg::angStream(int count_pair){
 switch(count_pair){
case 1: case 3: {
    elem22++;
    break;
    }
  case 2: case 4: {
    elem12++;
    break;
    }
  case 5:
    break;
  default:{
    Serial.print("error in angStream\n");
    break;
    }
  }
 }

void Hexaleg::lcheck(int count_pair){
 switch(count_pair){
  case 1: case 3: {
    if (elem22 == abs(elem23) && elem23 != 0){
      hStatus = HIGH;
      }
    break;
    }
  case 2: case 4: {
    if (elem12 == abs(elem13) && elem13 != 0){
      hStatus = HIGH;
      }
    break;
    }
  case 5:
break;
  default:{
```

```
Serial.print("error in angStream\n");
     break;
     }
   }
 }
void Hexaleg::reset(){
 elem12 = 0;
 elem13 = 0;
 elem22 = 0;
 elem23 = 0;
 hStatus = LOW;
 }
```

HexaLeg.hpp file:

```
#ifndef HEXALEG_H
#define HEXALEG_H
#include <Servo.h>
class Hexaleg {
 public:
   boolean hStatus;
   Hexaleg::Hexaleg(){}

   Hexaleg(Servo *s1, Servo *s2,int& Ang1, int& Ang2, int state1,
int state2, boolean s): firstServo{*s1}, secondServo{*s2},
elem13{Ang1}, elem23{Ang2}, side{s}, finitialState{state1},
sinitialState{state2}, hStatus {LOW} {}
   void finitState();
   void sinitState();
   void initialState();
   int readAng();
   void riseLeg();
   void lowerLeg();
   void rotRight();
   void rotLeft();
   void moveForward(int count_pair);
void moveBackward(int count_pair);
   void angStream(int count_pair);
   void lcheck(int count_pair);
   void reset();
   void turnOn();
   void reverse();
   void reverse_check();
 private:
   Servo& firstServo;
   Servo& secondServo;
   int finitialState;
   int sinitialState;
   const int elem11 = 1;
   int elem12 = 0;
   int& elem13;
   const int elem21 = 1;
   int elem22 = 0;
   int& elem23;
   boolean side;
};
#endif
```

HexaPair.cpp file:

```
include <Arduino.h>
#include "HexaPair.hpp"

void Hexapair::reverse_check(){
 fLeg.reverse_check();
 sLeg.reverse_check();
 tLeg.reverse_check();
 if (fLeg.hStatus == HIGH && sLeg.hStatus == HIGH &&
tLeg.hStatus == HIGH) pStatus = HIGH;
 }

void Hexapair::reverse(){
 fLeg.reverse();
 sLeg.reverse();
 tLeg.reverse();
 }

boolean Hexapair::readStatus(){
 return pStatus;
 }
void Hexapair::pcheck(int count_pair){
 fLeg.lcheck(count_pair);
 sLeg.lcheck(count_pair);
 tLeg.lcheck(count_pair);
 if (fLeg.hStatus == HIGH && sLeg.hStatus == HIGH &&
tLeg.hStatus == HIGH)
   pStatus = HIGH;
}
void Hexapair::turnOff(){
 pStatus = LOW;
 }
void Hexapair::turnOn(){
 pStatus = HIGH;
 }

void Hexapair::reset(){
 fLeg.reset();
 sLeg.reset();
 tLeg.reset();
 pStatus = LOW;
 }

void Hexapair::initialState(){
 fLeg.initialState();
 sLeg.initialState();
 tLeg.initialState();
 }
```

HexaPair.hpp

```
#ifndef HEXAPAIR_H
#define HEXAPAIR_H
#include "HexaLeg.hpp"
class Hexapair{
  public:
    boolean pStatus{LOW};
    Hexapair (Hexaleg& fl): fLeg{fl} {}
    Hexapair (Hexaleg& fl, Hexaleg& sl): fLeg{fl}, sLeg{sl} {}
    Hexapair (Hexaleg& fl, Hexaleg& sl, Hexaleg& tl): fLeg{fl},
sLeg{sl}, tLeg{tl} {}

    boolean readStatus();
    void initialState();
    void turnOff();
    void turnOn();
    void reset();
    void pcheck(int count_pair);
    void reverse();
    void reverse_check();
  private:
    Hexaleg& fLeg;
    Hexaleg& sLeg;
    Hexaleg& tLeg;

};



#endif
```

References

[1] J. L. Stephen Se, David Lowe, “Mobile Robot Localization and Mapping with Uncertainty using Scale-Invariant Visual Landmarks,” 2002.

[2] M. Betke and L. Gurvits, “Mobile robot localization using landmarks,” *IEEE Trans. Robot. Autom.*, vol. 13, no. 2, pp. 251–263, 1997, doi: 10.1109/70.563647.

[3] F. Shamsfakhr, B. Sadeghi Bigham, and A. Mohammadi, “Indoor mobile robot localization in dynamic and cluttered environments using artificial landmarks,” *Eng. Comput. (Swansea, Wales)*, vol. 36, no. 2, pp. 400–419, 2019, doi: 10.1108/EC-03-2018-0151.

[4] X. Chen, H. Sun, and H. Zhang, “A new method of simultaneous localization and mapping for mobile robots using acoustic landmarks,” *Appl. Sci.*, vol. 9, no. 7, 2019, doi: 10.3390/app9071352.

[5] R. St Pierre, W. Gao, J. E. Clark, and S. Bergbreiter, “Viscoelastic legs for open-loop control of gram-scale robots,” *Bioinspiration and Biomimetics*, vol. 15, no. 5, 2020, doi: 10.1088/1748-3190/ab9fa9.

[6] A. Whitman, G. Clayton, A. Poultney, and H. Ashrafiuon, “Asymptotic Solution and Trajectory Planning for Open-Loop Control of Mobile Robots,” *J. Dyn. Syst. Meas. Control. Trans. ASME*, vol. 139, no. 5, pp. 1–9, 2017, doi: 10.1115/1.4035169.

[7] F. Perez-Peña *et al.*, “Neuro-inspired spike-based motion: From dynamic vision sensor to robot motor open-loop control through spike-VITE,” *Sensors (Switzerland)*, vol. 13, no. 11, pp. 15805–15832, 2013, doi: 10.3390/s131115805.

[8] M. Schacher, “Optimal Open-Loop Feedback Control of Robots under uncertainty,” *Pamm*, vol. 10, no. 1, pp. 541–542, 2010, doi: 10.1002/pamm.201010263.

[9] G. Picardi, C. Laschi, and M. Calisti, “Model-based open loop control of a multigait legged underwater robot,” *Mechatronics*, vol. 55, no. March, pp. 162–170, 2018, doi: 10.1016/j.mechatronics.2018.09.006.

[10] J. Borenstein and L. Feng, “Measurement and Correction of Systematic Odometry Errors in Mobile Robots,” *IEEE Trans. Robot. Autom.*, vol. 12, no. 6, pp. 845–857, 1996, doi: 10.1109/70.544768.

[11] T. Abbas, M. Arif, and W. Ahmed, “Measurement and correction of systematic odometry errors caused by kinematics imperfections in mobile robots,” *2006 SICE-ICASE Int. Jt. Conf.*, pp. 2073–2078, 2006, doi: 10.1109/SICE.2006.315554.

[12] S. Maldonado-Bascón, R. J. López-Sastre, F. J. Acevedo-Rodríguez, and P. Gil-Jiménez, “On-board correction of systematic odometry errors in differential robots,” *J. Sensors*, vol. 2019, 2019, doi: 10.1155/2019/8269256.

[13] B. Barshan and H. F. Durrant-Whyte, “Inertial Navigation Systems for Mobile Robots,” *IEEE Trans. Robot. Autom.*, vol. 11, no. 3, pp. 328–342, 1995, doi: 10.1109/70.388775.

[14] D. Q. Khanh and Y. S. Suh, “Mobile robot destination generation by tracking a remote controller using a vision-aided inertial navigation algorithm,” *J. Electr. Eng. Technol.*, vol. 8, no. 3, pp. 613–620, 2013, doi: 10.5370/JEET.2013.8.3.613.

[15] Y. Rong, Y. Ko, X. Wang, and G. Shi, “Algorithm Research of Magnetometer Assisted Inertial Navigation System for Mobile Robots,” *NSENS 2019 - 2nd IEEE Int. Conf.*

Micro/Nano Sensors Al, Heal. Robot., pp. 92–96, 2019, doi: 10.1109/NSENS49395.2019.9293998.

[16] H. X. Liming Zhao, Jing Chen, Yi Zhang, Chuan Ye, Xiaodong Xu, "Research on method of vision navigation for mobile robot in unstructured environment."

[17] P. De Cristóforis, M. Nitsche, T. Krajník, T. Pire, and M. Mejail, "Hybrid vision-based navigation for mobile robots in mixed indoor/outdoor environments," *Pattern Recognit. Lett.*, vol. 53, pp. 118–128, 2015, doi: 10.1016/j.patrec.2014.10.010.

[18] J. M. Choi, S. J. Lee, and M. Won, "Self-learning navigation algorithm for vision-based mobile robots using machine learning algorithms," *J. Mech. Sci. Technol.*, vol. 25, no. 1, pp. 247–254, 2011, doi: 10.1007/s12206-010-1023-y.

[19] H. Quan, Y. Li, and Y. Zhang, "A novel mobile robot navigation method based on deep reinforcement learning," *Int. J. Adv. Robot. Syst.*, vol. 17, no. 3, pp. 1–11, 2020, doi: 10.1177/1729881420921672.

[20] W. B. Henrik Kretzschmar, Markus Spies, Christoph Sprunk, "Socially compliant mobile robot navigation via inverse reinforcement learning."

[21] A. Staroverov, D. A. Yudin, I. Belkin, V. Adeshkin, Y. K. Solomentsev, and A. I. Panov, "Real-Time Object Navigation With Deep Neural Networks and Hierarchical Reinforcement Learning," *IEEE Access*, vol. 8, pp. 195608–195621, 2020, doi: 10.1109/access.2020.3034524.

[22] S. Sendari, A. N. Afandi, I. A. E. Zaeni, Y. D. Mahandi, K. Hirasawa, and H. I. Lin, "Exploration of genetic network programming with two-stage reinforcement learning for

mobile robot," *Telkomnika (Telecommunication Comput. Electron. Control.*, vol. 17, no. 3, pp. 1447–1454, 2019, doi: 10.12928/TELKOMNIKA.V17I3.12232.

[23] M. E. Qazizada and E. Pivarčiová, "Mobile robot controlling possibilities of inertial navigation system," *Procedia Eng.*, vol. 149, no. June, pp. 404–413, 2016, doi: 10.1016/j.proeng.2016.06.685.

[24] G. Antonelli, S. Chiaverini, and G. Fusco, "A calibration method for odometry of mobile robots based on the least-squares technique: Theory and experimental validation," *IEEE Trans. Robot.*, vol. 21, no. 5, pp. 994–1004, 2005, doi: 10.1109/TRO.2005.851382.

[25] A. V. Rudyk, A. O. Semenov, N. Kryvinska, O. O. Semenova, V. P. Kvasnikov, and A. P. Safonyk, "Strapdown inertial navigation systems for positioning mobile robots—mems gyroscopes random errors analysis using allan variance method," *Sensors (Switzerland)*, vol. 20, no. 17, pp. 1–18, 2020, doi: 10.3390/s20174841.

[26] Y. Go, X. Yin, and A. Bowling, "Navigability of multi-legged robots," *IEEE/ASME Trans. Mechatronics*, vol. 11, no. 1, pp. 1–8, 2006, doi: 10.1109/TMECH.2005.863361.

[27] C. Zong, Z. Ji, and H. Yu, "Dynamic stability analysis of a tracked mobile robot based on human–robot interaction," *Assem. Autom.*, vol. 40, no. 1, pp. 143–154, 2019, doi: 10.1108/AA-10-2018-0156.

[28] M. Agheli and S. S. Nestinger, "Foot Force Based Reactive Stability of Multi-Legged Robots to External Perturbations," *J. Intell. Robot. Syst. Theory Appl.*, vol. 81, no. 3–4, pp. 287–300, 2016, doi: 10.1007/s10846-015-0233-z.

[29] M. Agheli and S. S. Nestinger, "Study of the Foot Force Stability Margin for multi-

legged/wheeled robots under dynamic situations," *Proc. 2012 8th IEEE/ASME Int. Conf. Mechatron. Embed. Syst. Appl. MESA 2012*, pp. 99–104, 2012, doi: 10.1109/MESA.2012.6275544.

[30] U. Asif, "Improving the navigability of a hexapod robot using a fault-tolerant adaptive gait," *Int. J. Adv. Robot. Syst.*, vol. 9, 2012, doi: 10.5772/50604.

[31] W. Chen, K. H. Low, and S. H. Yeo, "Adaptive gait planning for multi-legged robots with an adjustment of center-of-gravity," *Robotica*, vol. 17, no. 4, pp. 391–403, 1999, doi: 10.1017/S0263574799000958.

[32] M. Agheli and S. S. Nestinger, "Force-based stability margin for multi-legged robots," *Rob. Auton. Syst.*, vol. 83, pp. 138–149, 2016, doi: 10.1016/j.robot.2016.05.012.

[33] S. Inagaki, T. Niwa, and T. Suzuki, "Follow-the-contact-point gait control of centipede-like multi-legged robot to navigate and walk on uneven terrain," *IEEE/RSJ 2010 Int. Conf. Intell. Robot. Syst. IROS 2010 - Conf. Proc.*, pp. 5341–5346, 2010, doi: 10.1109/IROS.2010.5651324.

[34] D. Belter, J. Wietrzykowski, and P. Skrzypczyński, "Employing Natural Terrain Semantics in Motion Planning for a Multi-Legged Robot," *J. Intell. Robot. Syst. Theory Appl.*, vol. 93, no. 3–4, pp. 723–743, 2019, doi: 10.1007/s10846-018-0865-x.

[35] M. Jiang, G. Chi, G. Pan, S. Guo, and K. C. Tan, "Evolutionary gait transfer of multi-legged robots in complex terrains," *arXiv*, pp. 1–13, 2020.

[36] M. Jiang, Z. Huang, G. Jiang, M. Shi, and X. Zeng, "Motion generation of multi-legged robot in complex terrains by using estimation of distribution algorithm," *2017 IEEE Symp.*

Ser. Comput. Intell. SSCI 2017 - Proc., vol. 2018-Janua, pp. 1–6, 2018, doi: 10.1109/SSCI.2017.8285444.

[37] E. R. Yizhar Or, "Characterization of frictional multi-legged equilibrium postures on uneven terrains," [Online]. Available: https://doi.org/10.1177/0278364916679719.

[38] H. Zhang *et al.*, "A force-sensing system on legs for biomimetic hexapod robots interacting with unstructured terrain," *Sensors (Switzerland)*, vol. 17, no. 7, 2017, doi: 10.3390/s17071514.

[39] Umar Asif, "Virtual Reality to Simulate Adaptive Walking in Unstructured Terrains for Multi-Legged Robots," in *In Virtual Reality - Human Computer Interaction*, .

[40] G. Clark Haynes and A. A. Rizzi, "Gaits and gait transitions for legged robots," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2006, no. May, pp. 1117–1122, 2006, doi: 10.1109/ROBOT.2006.1641859.

[41] C. Ferrell, "A comparison of three insect-inspired locomotion controllers," *Rob. Auton. Syst.*, vol. 16, no. 2–4, pp. 135–159, 1995, doi: 10.1016/0921-8890(95)00147-6.

[42] W. A. Lewinger, "Insect-inspired, actively compliant robotic hexapod," *Sensors (Peterborough, NH)*, p. 128, 2005.

[43] O. Janrathitikarn and L. N. Long, "Gait control of a six-legged robot on unlevel terrain using a cognitive architecture," *IEEE Aerosp. Conf. Proc.*, 2008, doi: 10.1109/AERO.2008.4526240.

Biographical Information

Thong Nguyen was born in Ho Chi Minh city, Vietnam in 1996. He received his B.S degree of Mechanical Engineering in 2019 from University of Texas at Arlington and started his M.S of Mechanical Engineering in August 2019.

He has been working in number projects related in area of robotic engineering. He joined RBDSL laboratory under Dr. Alan Bowling in Fall 2019 and been involved in projects of hexapod. His research is related to the methods of fixing errors, controlling, and stability of hexapod.