SNAKE-LIKE ROBOT
(Programming Part)

WONG CHO GIAP

A project report submitted in partial fulfilment of the requirements for the award of the degree of Bachelor (Hons.) of Mechatronics Engineering

Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

April 2011

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature	:	_________________________
Name	:	____WONG CHO GIAP_____
ID No.	:	_______07UEB06710_______
Date	:	_______15 APRIL 2011_____

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"SNAKE-LIKE ROBOT"** was prepared by **WONG CHO GIAP** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor (Hons.) of Mechatronics Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature : _________________________

Supervisor : Prof. Dr. Wang Xin

Date : _________________________

SNAKE-LIKE ROBOT

ABSTRACT

This Project is to design a snake-like robot that can provide the locomotion as the real biological snake, and possessed the ability to cross over the obstacles with a certain height's limit, or find another alternative ways instead of climb over it if the height of the obstacles is over the limit. Snake-like robot is a biomorphic hyper-redundant robot that resembles a snake. The shape and sizes of the snake-like robot is depend on its own application, different application may required different sizes and shapes, since this project mainly target is to design a snake-like robot that can avoid the obstacles, so the snake-like robot is design to a moderate size with 8 segments, so that the snake-like robot can move flexible in the terrain that have a lot of obstacle. In order to make the snake-like robot function and move like a real biological snake, snake-like robot may construct of multiple joints which enable the snake-like robot to have multiple degree of freedom, which give it the ability to flex, reach and approach a huge volume in its workspace with infinite number of configurations. This mobility can enable the robot to move around in more complex environments. So, the application for this snake-like robot could be very useful in hard to reach places or hazardous environments. For the locomotion of this snake-like robot, it is move in a specific gait, which is a periodic of sine wave motion, just like a lateral undulation motion. Lastly, the special feature such as snake-like locomotion, ability to climb over obstacles or stair, estimate the height, making decisions, and able to remotely control are applied to the design.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

c_p	specific heat capacity, J/(kg·K)
h	height, m
K_d	discharge coefficient
M	mass flow rate, kg/s
P	pressure, kPa
P_b	back pressure, kPa
R	mass flow rate ratio
T	temperature, K
v	specific volume, m^3
α	homogeneous void fraction
η	pressure ratio
ρ	density, kg/m^3
ω	compressible flow parameter
ID	inner diameter, m
MAP	maximum allowable pressure, kPa
MAWP	maximum allowable working pressure, kPa
OD	outer diameter, m
RV	relief valve

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Background

Snake-like robot is a biomorphic hyper-redundant robot that resembles a snake. The shape and sizes of the snake-like robot is depend on its own application, different application may required different sizes and shapes, for example, for research and rescue purposed snake-like robot may required as thin as possible to enable it to do the tasks at narrow place. In order to make the snake-like robot function and move like a real biological snake, snake-like robot may construct of multiple joints which enable the snake-like robot to have multiple degree of freedom, which give it the ability to flex, reach and approach a huge volume in its workspace with infinite number of configurations.

For the locomotion of the snake-like robot, there still don't have any snake-like robot can completely mimic the locomotion of real snake, most of the snake-like robot moves by refer to a specific gait, where a gait is a periodic mode of locomotion, for example a gait maybe side winding. There have several type of locomotion for a snake-like robot, which included the movement using the wheels, 'legs', or without both of the wheels and leg but just slide, glide, and slither. Below show the advantages of the serpentine locomotion over the locomotion using wheels or legs:-

- Stability- where the serpentine locomotion have a better stability than wheels and legs, this because the serpentine robot will never fall, even if a free fall occurs, the serpentine robot will survive better than most mobile devices, because the potential failure points like the connections between the body and wheels or leg do not exist.

- Terrainability- means the ability of moving in a rough terrain, if compare the serpentine robot to the robot using wheels or legs, the serpentine robot have better terrainability than the latter, for example, wheels or legs will stuck at a hole but this won't happen to a serpentine robot.

However, there also have several disadvantages of the serpentine robot over the locomotion of the wheeled or legged devices:-

- Degree of freedom- a large number of actuators is required if compare with the wheeled or legged device to help the serpentine robot to subtend the various curves, this means the serpentine robot may have a large number of degree of freedom, where this will introduce reliability problems, that is if failure occur on one of the actuators, then the robot with large number of units have a higher chance of having any unit fail.
- Speed- The speed for a serpentine locomotion robot is fairly slow if compare to the real snake.

For the application of the snake-like robot, it may include several fields such as the medical, exploration, routing, and military. For medical, people had applied the snake-like robot technologies into the surgery, for example, the snake-like robot is operating in the narrow throat region, to make incisions and tie sutures with greater dexterity and precision.

In exploration, the snake-like robot is suitable to explore the unpredictable environment, this may due to the ability of snake-like robot to distribute its mass over a large area for support so that if the footing is gives way, self-support between secure points enables continued operation.

For routing purpose, such as wiring through passages behind existing walls or through the pipes, these tasks involve long reaches and awkward position through the pipes, so the flexible snake-like robot will be required to make the work easy by maneuver through crowded plenums and pull the initial light weight tapes which will use to pull the actual cables.

In military, one of the example maybe the snake-like robot manufacture by Isareal, where the robot can sneak through cracks and into buildings to send back sound and video of enemy movements or even plant explosives.

1.2 Aims and Objectives

The objective of this project is to design a snake-like robot, which can either move by making decision itself in an unknown terrain (inspection) or move remotely with the assist of camera. Besides that, we are also required to program the snake-like robot so that it can move similar as the biological snake. The project is classified, which are the:-

- Mechanical part

  For mechanical part, the body or other mechanical parts can suit to the locomotion of the snake-like robot have to be ensure, so that it can move in 2-axis (the xy-plane), able to overcome the obstacle and move on uneven surface, and the weight of the chosen material is important for the locomotion as well.

- Electronics part

  For electronics part, the effective voltage regulator for supplying efficient power to the motor and microprocessor have to be ensure, and solving the battery life issue being considered inefficient when supplying different power ratings by using linear voltage regulator.

- Programming part

  For programming part, programming on the snake-like movement for the snake-like robot need to be done, and with the vision assist of the camera, the snake-like robot will able to decide the possible solutions to overcome the obstacle that it face, and also able to respond to the feedback of the controlling sensors like angle sensor.

  In Programming part, the scopes are set as below:-

  -The snake-like robot able to provide snake-like locomotion.
  -The snake-like robot can make the decisions itself when facing problems (obstacles).
  -The distance can be detect with the assist of sensors.
  -The snake-like robot can lift at a certain height so that it can climb stair or climb over the obstacles.
  -The snake-like robot can remotely control by the user.

CHAPTER 2

LITERATURE REVIEW

Snake-like robots are multi-segmented devices. Based on their physical structure and design, these robots could have great mobility in their movements. This mobility can enable the robot to move around in more complex environments. The application of these kind of robots could be very useful in hard to reach places or hazardous environments, this is one of the reason that make the snake-like robots playing important role in our life and had been utilized in many fields like research and rescue, military, inspection and others.

2.1 Design and research of a new structure rescue snake robot with all body drive (rope drive system)

After earthquake, many people are buried under ruins. Rescue is very important in the first 48 hours, most survive are dead after 48hours. Snake robot has slime body and can reach the narrow crevices that people can't do, so is a very useful tool in climbing into ruins to detect people.(GAO etc, 2008)

This kind of snake robot is using the rope drive system to move like a snake. The snake robot have 11 bodies and 10 joints, with the head and tail body have camera and communicator but without the drive system, where the drive systems are on the middle parts of the snake robot, which contain 9 segments. (GAO etc, 2008) Each segment contain a CPU, where the main CPU is located at the tail, so that the data such as velocity, forces, and angle value can send to the main CPU to process since the head and middle parts are move first. The main material for the body part is using the engineering plastic, which diminish the total weight, and increase the mobility of the snake robot.

Each segments in the middle part have a rope drive system, and fixed with many of directive wheels on the edge of the segment body, which used to move the rope that surrounding the segment body, below show the construction of the rope drive system:-

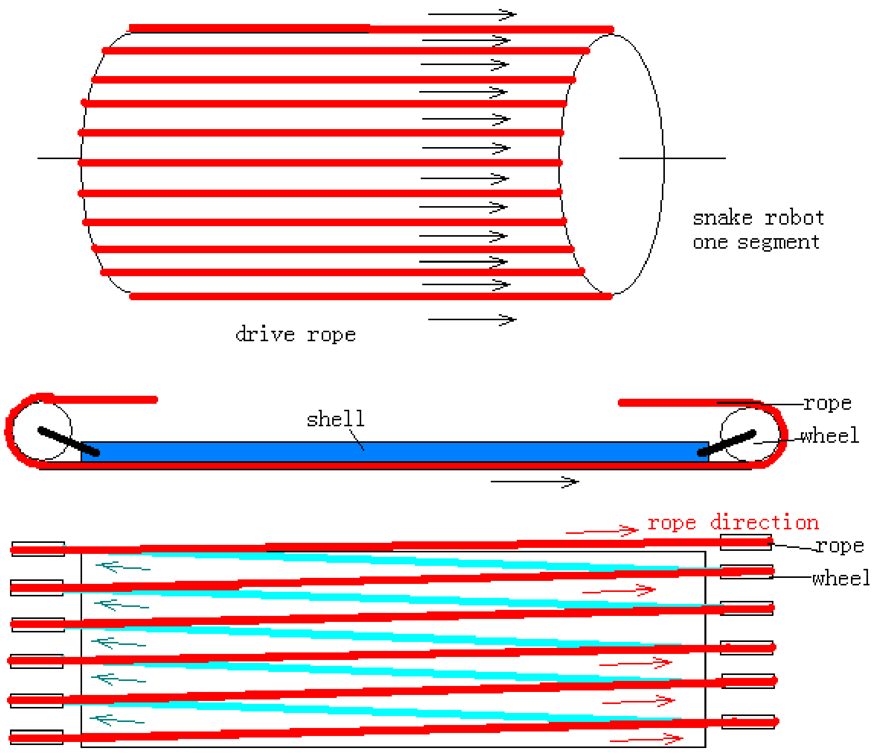


Figure 2.1: Rope drive system on each segment.

The rope is drive by the drive friction wheel with a press friction wheel contact on the rope, where the drive friction wheel is drive by the motor in each segment.

The advantage the all body drive robot with rope drive system over the other snake robot is the ability to move the robot with any segment/s which touches on the ground, this is because the drive force on each segment of the snake robot.

One of the disadvantages of the rope drive system is that, the maintenances of the snake robot will be complicated, this is because the construction of this system contains of many ropes and directive wheels, which malfunction of one segment will lead to a long time in troubleshooting and repair. Besides that, the wheels may also require the lubrication as well.

2.2 Locomotion mechanism of snake-like robot

Snake move by pushing their body against environment. In order to achieve the real snake's locomotion, the snake robot must generate a lateral undulation, where lateral undulation is a sequence of left-right wave movements. Besides that, the co-coordinate between the adjacent segments and the robot performance in case there have any segment of the snake robot is failure are the important issues in designing the locomotion for the snake robot. (Hua etc ,2001)

Snake robot is driven in a close circuit, which means the processor in the snake robot will receive any feedback signals from the sensors and respond to the feedback by making a new decision for the snake robot. Below show the control configuration of the snake robot:-

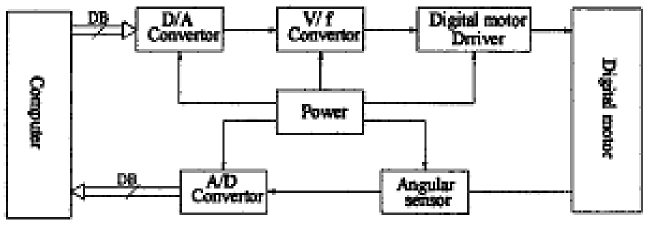


Figure 2.2: The control configuration of the snake robot.

The locomotion mechanism of the snake robot can be analyzed in moving forward, backward and steering. For moving forward and backward, a mathematical model can be used to analyze the locomotion mechanism:-

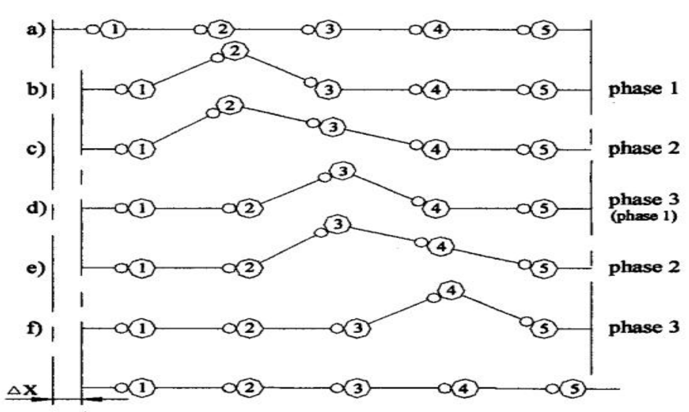


Figure 2.3: The locomotion mechanism of moving forward and backward.

From the figure 2.3, the small circles represent the motor and the big circles represent the big gear, where the straight lines represent the segments for snake robot. The wave is produced at first phase by moving the gear 1 and 2, at the same time, the end of the snake robot had been move forward with the distance equals to Δx. The wave will then continue propagate to the gear 3 and 4 and last to the gear 5 which will push the head of the snake robot forward with the same distance Δx.

The locomotion mechanism of moving forward and backward can be used as the reference in design the snake like robot, but the direction of the gears can be change from moving upward and downward to left and right to produce the lateral undulation, where the motors and big gears in the figure2.3 can be replaced by the servo motor, and the left right locomotion concept can be achieved by programming the servo motor.

2.3 Toroidal skin drive for snake robot locomotion

Most robotic snake designs have fallen into one of two categories, which are those that use undulation of the internal structure (the spine) to move the body along and those that use some sort of surface mounted wheels or tracks to move the body, where toroidal skin drive is an alternative approach for snake robot locomotion.(McKenna etc, 2008) The toroidal skin drive system is actually the system that locomote the snake robot by moving the skin of the robot with assist of the drive unit on the skin. Below show the constructions of the toroidal skin drive system:-

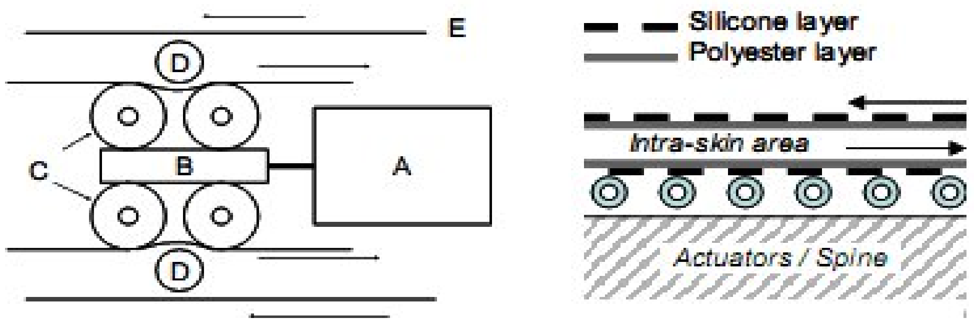


Figure 2.4: The toroidal skin drive mechanism.

From the figure, the A represent the motor, B represent the worm, C represent the worm gears which also act as the drive wheels, D is the tension ring captured inside the skin in order to maintain pressure between the skin and the gear sets, where the skin is represented by the E. The skin have divided into inner part and outer part, but they are wraps together at the end, where the motion of the skin is control by the gear sets which power by the motor, as the inner part is moving, the outer part is moved as well but is in opposite direction, and this may propel the snake robot to move forward or backward due to the friction between the ground surface and the outer skin of the snake robot. However, the skin drive system only provide the rectilinear motion, so the angular actuator is required to help the snake robot to maintain body configuration. This may help in simplify the kinematics of motion as well. The angular actuator consists of the gears and motors.

The main advantage of this kind of drive system is that similar with the rope drive system with every point on the surface of the robot provides forward locomotion. (McKenna etc, 2008)

2.4 Pulse Width Modulation for programming servo motor.

Pulse width modulation (PWM) can be very useful for controlling analog circuits with a processor's digital outputs. PWM is employed in a wide variety of applications, ranging from measurement and communications to power control and conversion. The PWM also very useful in programming the servo motor, because the PWM can position the shaft of the motor to a specific angle, this may achieve by send an ordinary logic square wave to the servo at a specific wave length. Below show the time vs angle graph:-

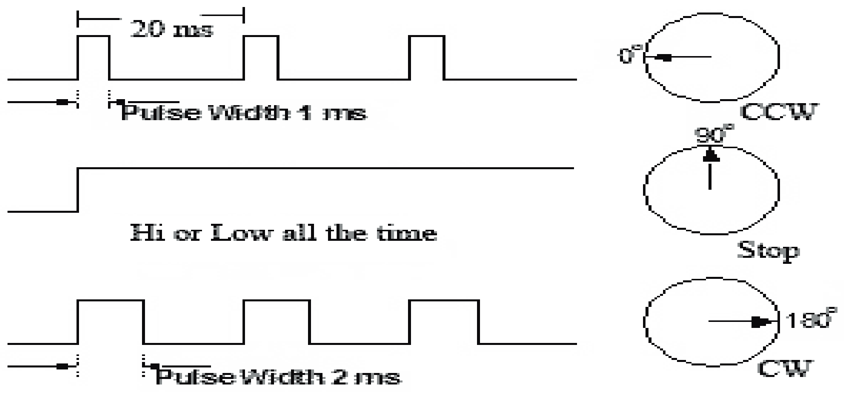


Figure 2.5: The chart of time vs angle for PWM control.

From the example show above, the angle is on 0° when the input pulse width is on 1ms, and change to 90° when there is HI or LOW all the time in the interval of 20ms, this is the neutral position for the servo, but there also have another condition that can make the servo motor to stop at this neutral position, that is when the input pulse width is equals to 1.5ms which is the value at middle of the pulse width for 0° and 180°, where a 2ms pulse width is needed to turn it from any angle to the 180°.

This PWM method may be very useful in design the locomotion of snake-like robot, this is because the sine wave motion can be produce by programme the servo motor to turn at specific angle at specific time.

2.5 Summary for literature reviews.

Table 2.1: Summary of literature reviews.

Year of publish	Author/Journal	Technology	Analysis
2008	Junyao Gao; Xueshan Gao; Wei Zhu; Jianguo Zhu; Boyu Wei; , "Design and research of a new structure rescue snake robot with all body drive system,"	• Each segment contain one CPU • Main CPU locate at the tail. • Many directive wheels fixed at both ends of each segment and surrounded with the rope. • Use the friction between the rope and the ground to move when the motor move the rope.	• Advantage: The drive force on every segment. • Disadvantages: maintenances of the snake robot will be complicated.
2001	Hua Liu; Guozheng Yan; Guoqing Ding; , "Research on the locomotion	• Closed loop, means response to the feedbacks from the sensors. • Moving forward and backward mechanism.	• The method of moving forward and backward can change from moving upward to a sine waveform, just right and left.

	mechanism of snake-like robot,"		
2008	McKenna, J.C.; Anhalt, D.J.; Bronson, F.M.; Brown, H.B.; Schwerin, M.; Shammas, E.; Choset, H.; , "Toroidal skin drive for snake robot locomotion,"	• Toroidal skins drive system. • The inner part of skin is wraps together with the outer part at the end of the snake robot. • The tension rings captured inside the skin to give the pressure between the skin and the gear. • Once gears rotate, it rotates the skin as well.	• Advantages: simplify the kinematics of motion. • Drive force at every segment. • Disadvantage: only provide the rectilinear motion, unless angular actuator is used.
2010	Website information: http://mcu-programming.blogspot.com/2006/09/servo-motor-control.htm "Pulse Width Modulation for controlling servo motor."	• Position the servo motor shaft by send the pulse with certain width to a certain angle position.	• Can be apply to the locomotion of snake-like robot, so that the robot can provide a sine wave motion by keep changing the angle position of the motor shaft.

CHAPTER 3

METHODOLOGY

3.1 Project Overview

The snake-like robot will be separated into three different parts of tasks, namely the mechanical part, the electronics and electrical part, and the programming part. Thus, a team is formed of three person with each person takes a part.

This report takes the tasks of programming part. The programming part will include the control for locomotion of the snake-like robot by programming the servo motor, response to the sensors feedback, response to user command, and able to lift the body in particular applications like stair climbing and cross over the abstacles.

Below is the flow graph for the overview of snake-like robot design project:-

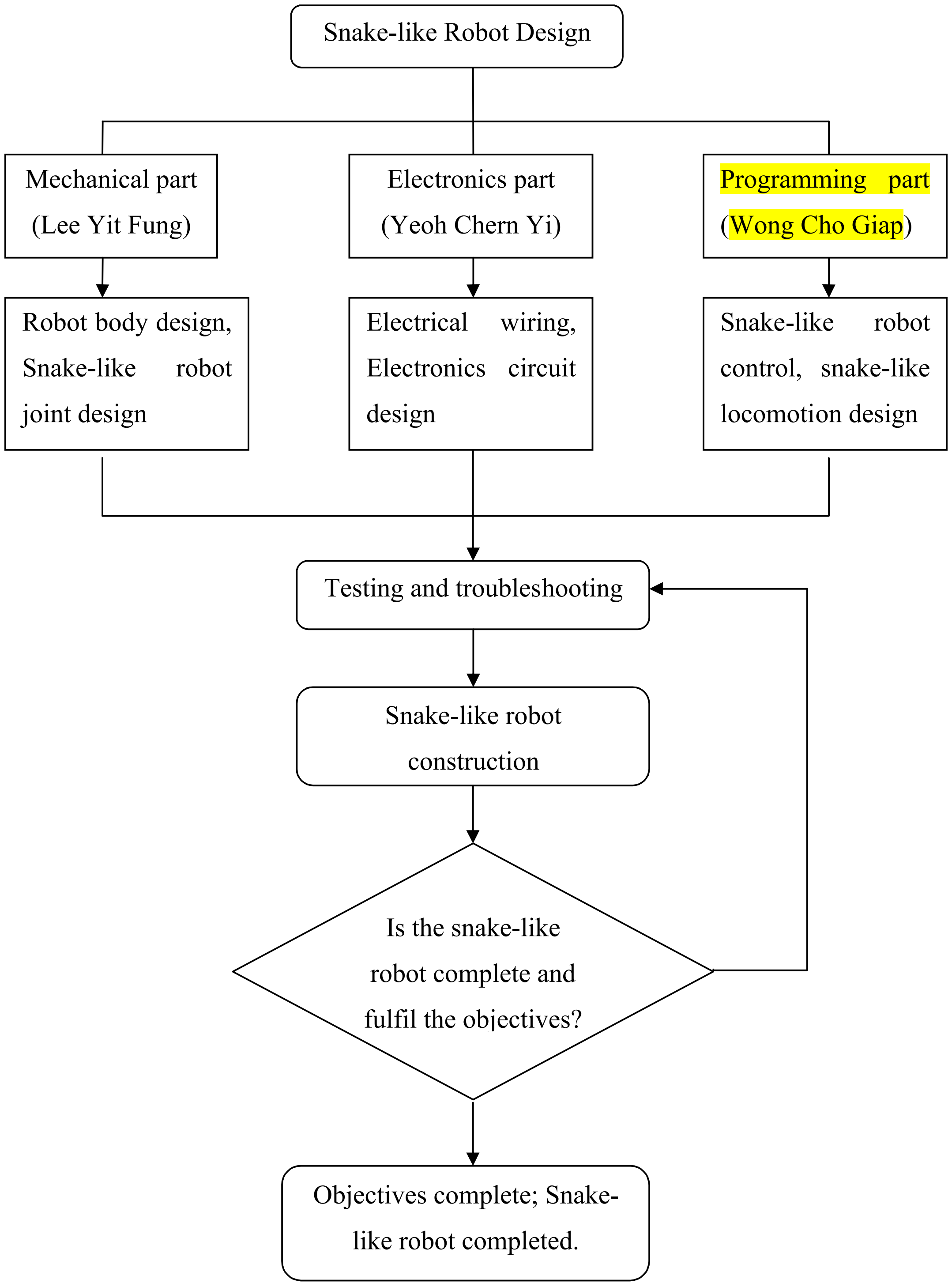
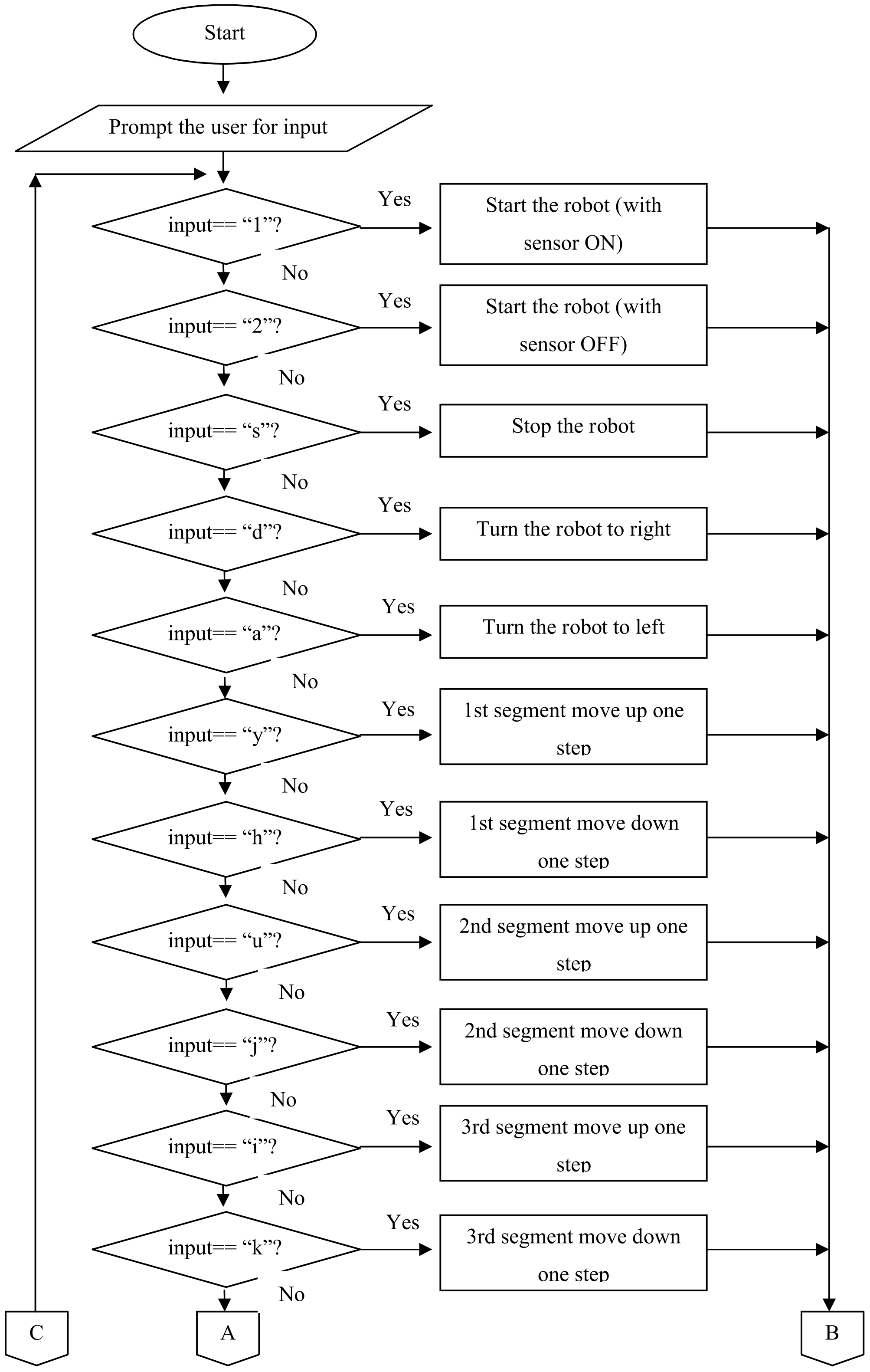

Snake-like Robot Design
Mechanical part (Lee Yit Fung)
Electronics part (Yeoh Chern Yi)
Programming part (Wong Cho Giap)
Robot body design, Snake-like robot joint design
Electrical wiring, Electronics circuit design
Snake-like robot control, snake-like locomotion design
Testing and troubleshooting
Snake-like robot construction
Is the snake-like robot complete and fulfil the objectives?
Objectives complete; Snake-like robot completed.

Figure 3.1: The flow graph for the snake-like robot design project.

3.2 Flow chart

3.2.1 Flow chart for the control of snake-like robot.


Start
Prompt the user for input
input== "1"?
Yes
Start the robot (with sensor ON)
No
input== "2"?
Yes
Start the robot (with sensor OFF)
No
input== "s"?
Yes
Stop the robot
No
input== "d"?
Yes
Turn the robot to right
No
input== "a"?
Yes
Turn the robot to left
No
input== "y"?
Yes
1st segment move up one step
No
input== "h"?
Yes
1st segment move down one step
No
input== "u"?
Yes
2nd segment move up one step
No
input== "j"?
Yes
2nd segment move down one step
No
input== "i"?
Yes
3rd segment move up one step
No
input== "k"?
Yes
3rd segment move down one step
No
C
A
B

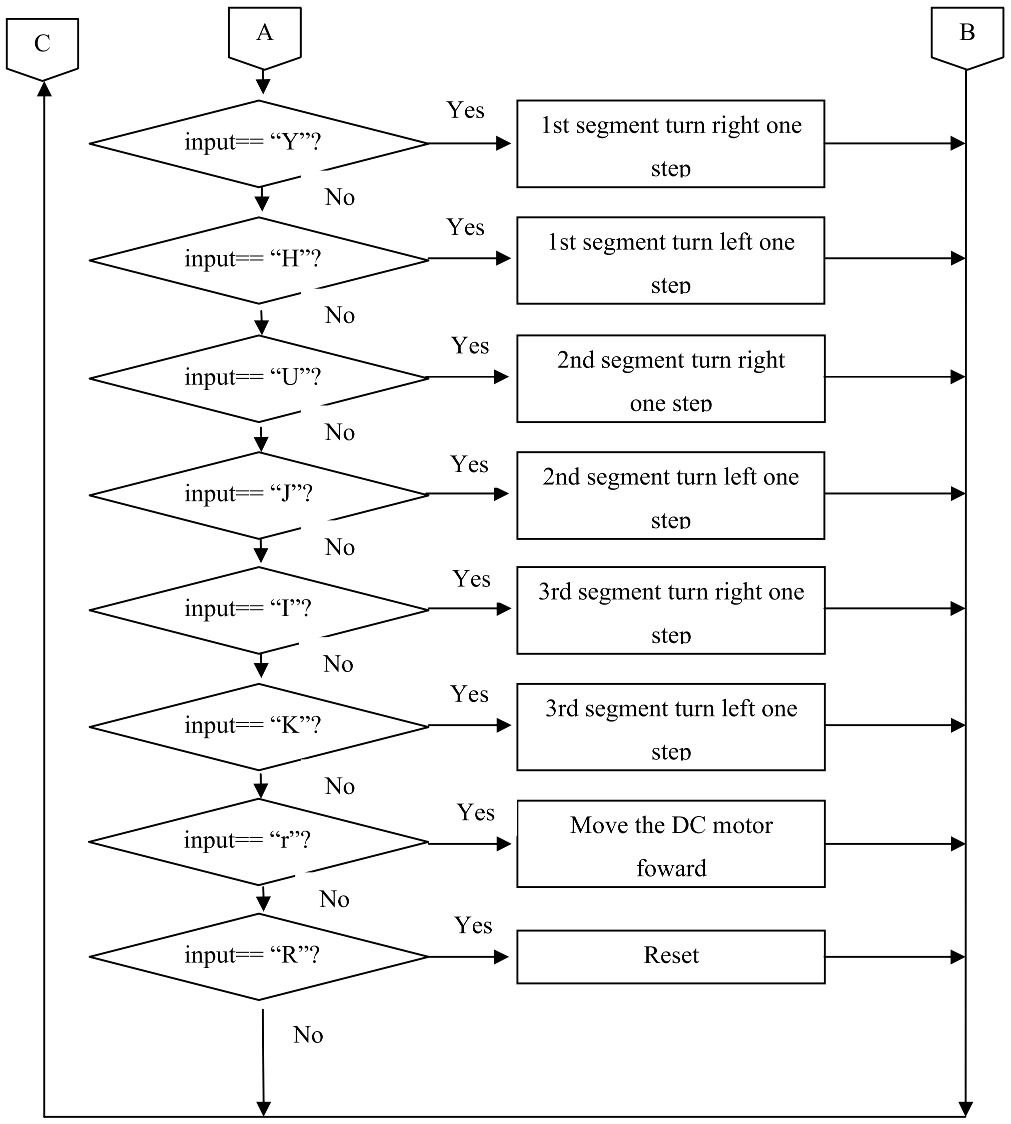
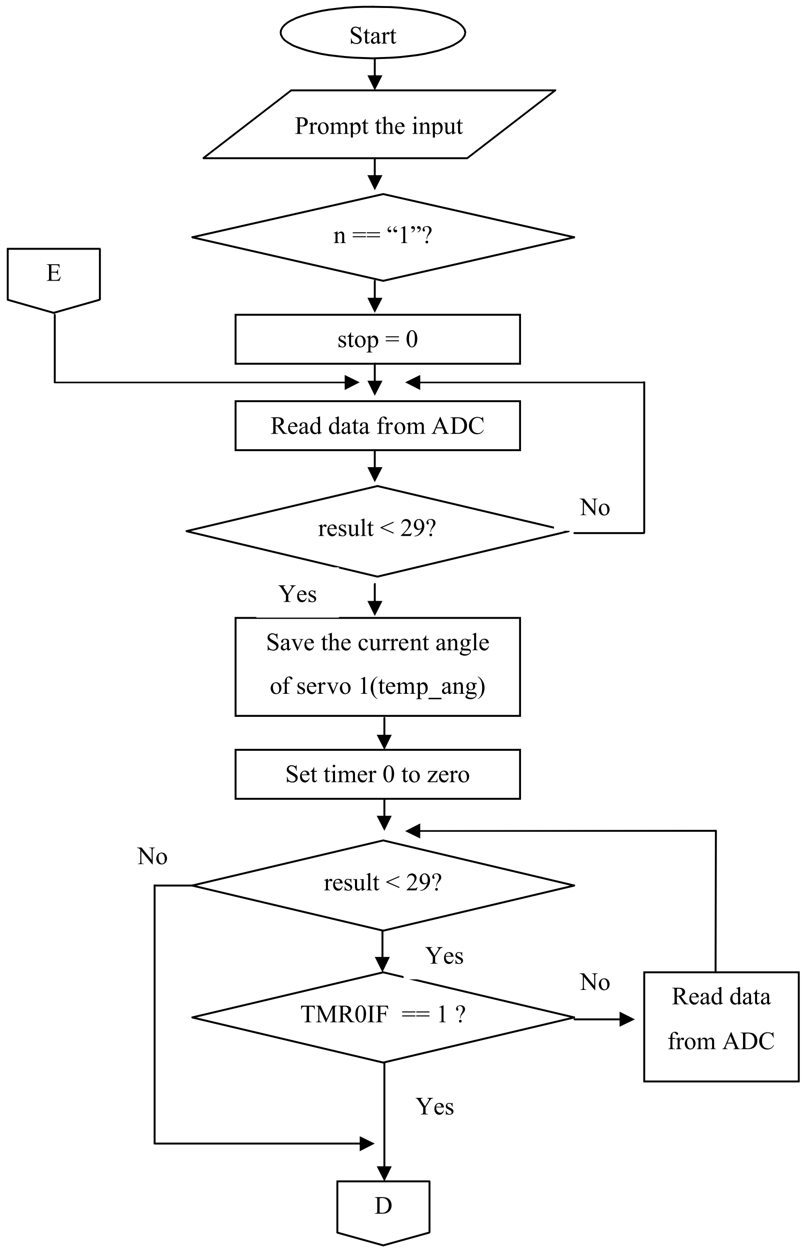

C
A
B
input== "Y"?
Yes
No
1st segment turn right one step
input== "H"?
1st segment turn left one step
input== "U"?
2nd segment turn right one step
input== "J"?
2nd segment turn left one step
input== "I"?
3rd segment turn right one step
input== "K"?
3rd segment turn left one step
input== "r"?
Move the DC motor foward
input== "R"?
Reset

Figure 3.2: The flow graph for the control of snake-like robot.

The figure above has shown control flow of the snake-like robot. Once the snake-like robot is power up, the PIC will send the signal to the computer via Xbee wireless to prompt inputs from the user. There consist of 19 cases for the switch command, where the snake is initially stop and all servos are move to center to get ready for the command. For 19 of the cases, only the inputs "1" and "2" can start the snake-like robot to move in snake locomotion, while "s" is to stop the snake like robot, "d" and "a" are to change the robot direction to the right and left respectively, "r" is the input command for the robot to command the DC motor move forward to the direction that the robot positioning, where the "R" is the reset button for the robot to initialize all the data . Other than those general control's inputs, the command's inputs to move each segment of the robot to up, down, right and left are introduce to the robot as well, so that the movement of the robot can manually control by the user. The flow is keep repeat until the power is off.

3.2.2 Flow chart for main flow of the snake-like robot (Automatically).



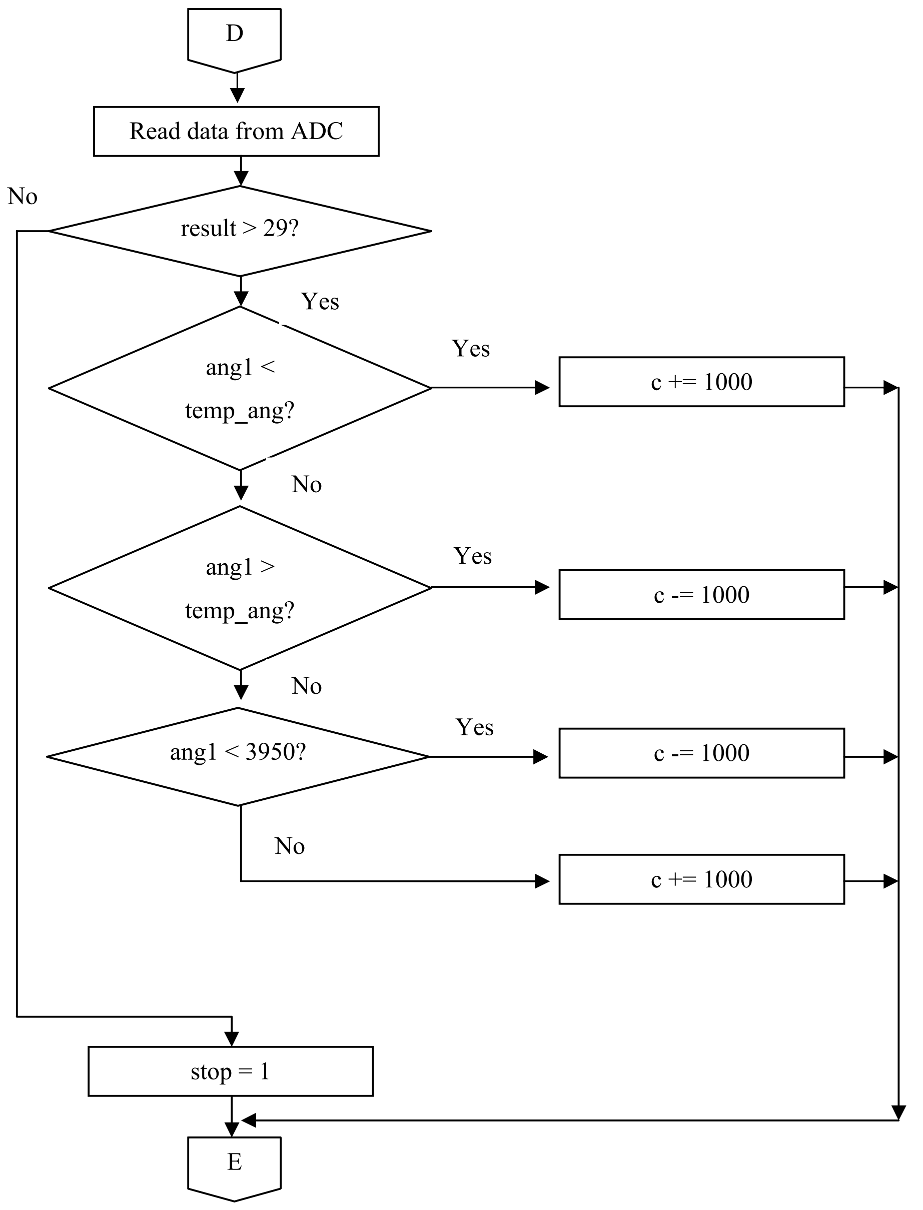


Figure 3.3: The main flow of snake-like robot. (Automatically)

The figure above is the main flow of the snake-like robot with the range finder sensor activated, so that the robot can make the decision itself to avoid the obstacles. This mode is activated when the user send the input “1”, where most of the time in the flow, the PIC keep getting the data from the ADC, which convert the info of the sensor from voltage (analogue) to binary number (digital). With the information (distance) from the sensor, the robot can differentiate the obstacle, and determine the direction to move in the next step. The method that the robot used to determine will be discussed in below. In order to make the robot to be controlled in semi-auto mode, the sonar (range finder) sensor can be turn off by switching the mode to “2”, so that the differentiate method can be ignore by the PIC

3.2.3 Flow chart for High-priority and Low-priority interrupt in the program.

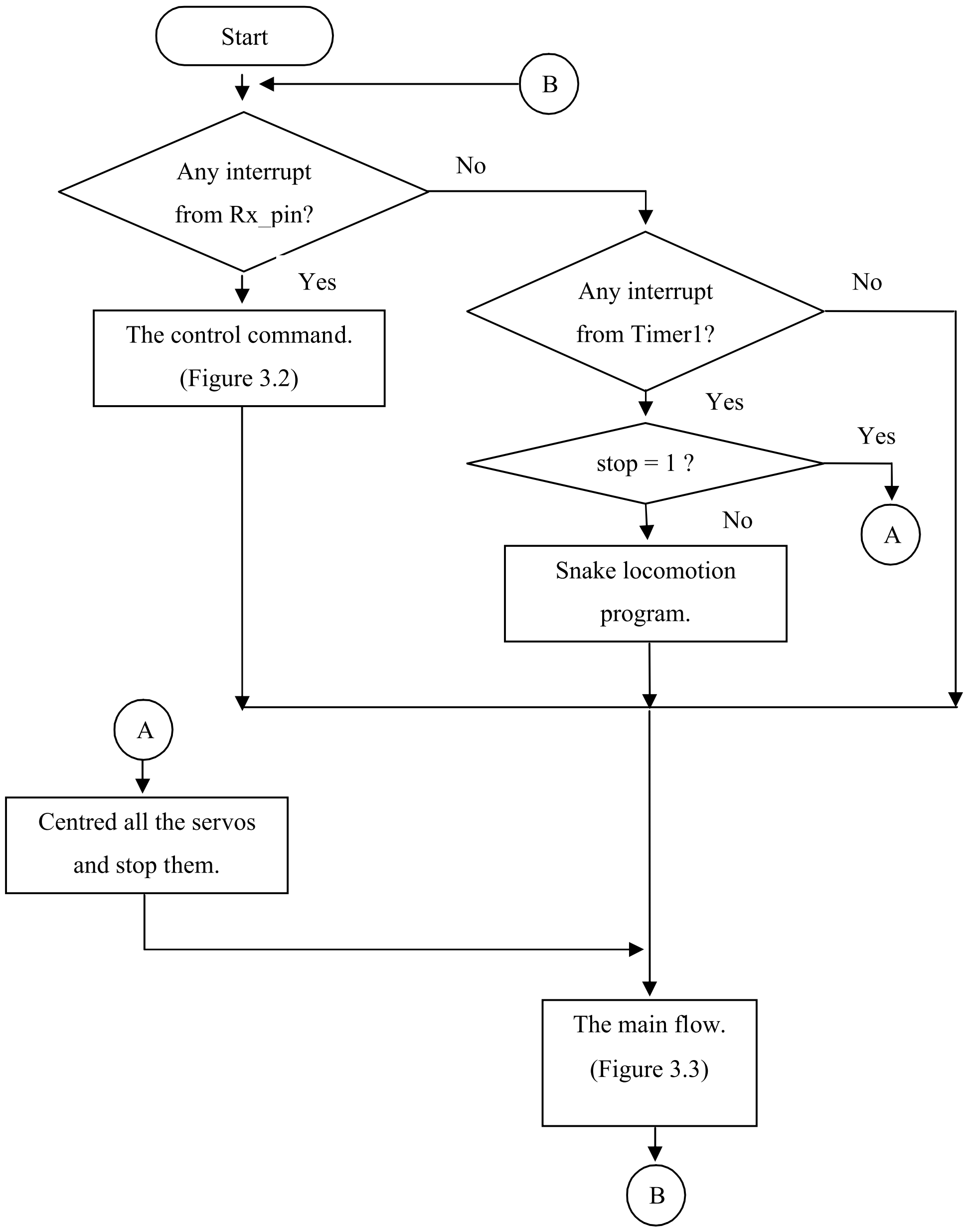

Start
B
Any interrupt from Rx_pin?
No
Yes
The control command. (Figure 3.2)
Any interrupt from Timer1?
No
Yes
stop = 1 ?
Yes
No
A
Snake locomotion program.
A
Centred all the servos and stop them.
The main flow. (Figure 3.3)
B

Figure 3.4: The flow chart for the interrupt service routine of the robot.

Figure above are the flow chart for the interrupt service routine in the program of the snake-like robot. The interrupts had divided into high priority and low priority for the microcontroller to differentiate which interrupt service routine to process first once there have two interrupts arise in the same time. From the flow chart, it shows that the interrupt for receive pin of the serial communication (Rx_pin) is set to high priority interrupt, while the Timer 1 is set to low priority.

3.3 Programming language and microcontroller

In this project, the C language is chosen as the programming language instead of Assembly language due to the several major reasons:

- Much easier and less time consuming to write program using C than using Assembly.
- C is easier to modify and update.
- Can use the code available in function libraries.
- C code is portable to other microcontrollers with little or no modification.

However, the hex file produced by using C is much larger than using Assembly language. Due to this problem, the 40-pins microcontroller PIC 18F4520 has been chosen. The PIC 18F4520 allowed the program memory up to 32 kilobytes, which are more than enough to load the command program of the snake like robot in this project. Besides that, the features of this PIC such as provided 36 inputs/ outputs pin, 13 channels of analog-to-digital converter, 2 CCP modules, and 3 timers. This project using 40-pins PIC instead of other, because the snake robot required a lot of input and output pins, the table below shows the I/O pins used in the snake like robot:

Table 3.1: Input and Output pins used in snake like robot.

Inputs/Outputs Pin	Description
RA5 (AN4)	Used as the analog-to-digital converter channel (sensor).
RA7 (OSC1)	For the external clock source (20 Mhz crystal is used).
RA6 (OSC2)	For the external clock source (20 Mhz crystal is used).
RC2 (CCP1)	PWM output for DC motor.
RD1	The output indicator LED for sensor.
RB1-7	The output signals (duty cycle) for servo 1-7 respectively.
RD7	The output signals (duty cycle) for servo 8.
RB0	For switch (To start the robot without using wireless device).
RC7 (RX)	Receive information from user through wireless device.
RC6 (TX)	Transmit information to user through wireless device.

3.4 The snake equation

There are four types of movement for snake, which are the serpentine movement, rectilinear movement, concertina movement, and side-winding movement. However, for this project, only the serpentine movement is applied as it is the most frequently used form of snake locomotion by most of the snake.

For the serpenoid curve analysis, it is introduced by Hirose. In this project, the serpenoid curve is used as the basic body movement of the snake robot. While, the serpenoid curve can be modeled with the following equations (Chang & Chen 2008):

$$x(s) = \int_0^s \cos(\zeta_\sigma) d\sigma \tag{3.1}$$

$$y(s) = \int_0^s \sin(\zeta_\sigma) d\sigma \tag{3.2}$$

$$\zeta_\sigma = a\cos(b\sigma) + c\sigma \quad (3.3)$$

Where the parameter *a*, *b*, and *c* are the variables to determine the shape of the serpentine motion. The variable *a* determines the degree of the serpentine motion, *b* determines the number of phase of the serpentine motion, and *c* determines the direction of the serpentine motion.

The serpentine curve can be applied into the snake like robot by changing the relative angles between the segments using the following formula with the number of segments (n=4):

$$\phi_i = \alpha sin(\omega t + (i-1)\beta) + \gamma, (i = 1, ..., n-1) \quad (3.4)$$

Where α, β, and γ are the parameters depend to the variables a, b, and c respectively. The relationships are shows below:

$$\alpha = a\left|\sin\left(\frac{\beta}{2}\right)\right| \quad (3.5)$$

$$\beta = \frac{b}{n} \quad (3.6)$$

$$\gamma = -\frac{c}{n} \quad (3.7)$$

The equations (3.3), (3.4), (3.5), and (3.6) are applied into the programming code, where the default values of each variable are set as below:

$$a = 45 \text{ degree} \quad (3.8)$$

$$b = 3\pi \quad (3.9)$$

$$c = 0 \quad (3.10)$$

$$n = 4 \qquad (3.11)$$

With the values of each variable show above, the parameters α, β, and γ can be calculated by the microcontroller as well. Since the snake like robot in this project only has 4 segments, hence, it only requires three equations for the joints (three joints). Below shows the equations for the angle of each joint of the snake like robot in programming code:

```
ang1=(alpha*sin(t)+gamma+3950);
ang2=(alpha*sin(t+1*beta)+gamma+3950);
ang3=(alpha*sin(t+2*beta)+gamma+3950);
```

The number 3950 in the equations represent the decimal value for timer to produce the duty cycle that position the servo motor to 90 degree. In order to make the reference axis centred at 90 degree, the number 3950 is added into the equations. Besides that, the gamma in the equations is used to determine the direction of snake like robot. If gamma is decreasing, the snake will turn to the left direction, or turn to the right direction when the gamma is increasing.

3.5 The polling and interrupt method

In this project, both polling and interrupts method are applied. The polling method is used to transmit data to the user, while the interrupts method is used by the timer (Timer 1) to produce the particular duty cycle for servo motors and to receive data from user via serial communication. In polling method, the microcontroller will continuously monitors the status of the TXIF, when the transmit interrupt's flag is low, means the data in TXREG already transmitted to the user via the wireless device, so it will insert the next data to the TXREG again, and the polling will keep repeated until all of the data are transmitted to the user. In interrupt method, whenever the wireless device (to receive data from user) or servo motors needs the microcontroller's service, they will notify it by sending an interrupt signal. Upon receiving the interrupt signal, the microcontroller stops whatever it is doing and serves the device. The interrupt service routine (ISR) are required in interrupts method, because the microcontroller will go to the particular ISR with respect to which interrupt signal it received, and process the command inside the ISR. Below is the program to tell the microcontroller where to go when it receives the interrupt signal either from received interrupt (serial communication) or Timer 1 interrupt:

```
#pragma code Rx_int=0x0008          //interrupt vector for high-priority interrupt
void Rx_int(void)
{
        _asm
                GOTO chk_rx_isr     //position the microcontroller to service the
        _endasm                     // receive's ISR
}

#pragma code Timer_int=0x0018       //interrupt vector for low-priority interrupt
void Timer_int(void)
{
        _asm
                GOTO chk_timer_isr  //position the microcontroller to service the
        _endasm                     // timer's ISR
}

#pragma interruptlow chk_timer_isr
void chk_timer_isr(void)            //timer ISR
{
        if (PIR1bits.TMR1IF==1)
        {
                if (stop == 1)
                        stop_isr();
                else
                        T1_isr();
        }
}

#pragma interrupt chk_rx_isr
void chk_rx_isr(void)               //receive's ISR
{
        if(PIR1bits.RCIF==1)
                Rx_isr();
}
```

As the programming code shows above, the snake robot is a two-level priority interrupt system with the IPEN (interrupt priority enable) bit is set to high. The purpose to classified the interrupts into high and low priority is to helps the microcontroller to determine which ISR to process first when there have two or more interrupts activated at the same time, or when the microcontroller is executing an ISR of an interrupt and another interrupt is activated. In such case, the microcontroller will choose to process the high-priority interrupt first. Below is the setting code for interrupts:

```
PIE1bits.RCIE=1;                //enable the receive interrupt bit
RCONbits.IPEN=1;                //enable the interrupt priority
IPR1bits.TMR1IP=0;              //set timer1 interrupt to low priority
PIR1bits.TMR1IF=0;              //clear the timer1 interrupt flag
PIE1bits.TMR1IE=1;              //enable the timer1 interrupt bit
INTCONbits.GIEH=1;              //enable all high priority interrupt
INTCONbits.GIEL=1;              //enable all low priority interrupt
```

From the code, both interrupts for Timer 1 and received interrupt are enabled, and the received interrupt (serial communication) is set to high-priority, while the Timer 1 is set to low-priority.

3.5.1 The high priority interrupts (received interrupt).

The received interrupt (serial communication) is set to high-priority interrupts. Thus, when the interrupt flag of the received interrupt, RCIF is raised after the entire frame of data, including the stop bit, is received, the microcontroller will be directed to the rom location at 0x0008 and process the ISR of the received interrupt. As high-priority interrupt, the microcontroller will process the data received from the user first and stop executing the current program even for the ISR of timer (low-priority).

The received interrupt is used to process the information from the user via the wireless device, so that the user can remotely control the snake like robot by sending the data to the microcontroller. The details of the process for serial communication will be discussed on the sub-chapter for serial communication.

3.5.2 The low priority interrupts (Timer 1 interrupt).

The Timer 1 interrupt is low-priority interrupt, it will direct the microcontroller to the ISR for low priority interrupts once the TMR1IF is raised to indicate that the Timer 1 is overflowed. However, the timer's interrupt will not be process until the finish process of ISR for high-priority interrupts by the microcontroller, due to the lower priority. Timer 1 is used to determine the duty cycle of the signal to servo motors over time. The details of using Timer 1 to control the position of servo motors will be discussed on the sub-chapter for timers.

3.6 Serial data communication for remotely control (via wireless device).

In order to manually or semi-automatically control of snake-like robot with the assist of user command, a wireless device is used to communicate between the microcontroller and computer (user interface). The wireless device used in this project is two modules of xbee starter kit, where one module will be installed to computer via USB communication, and another will be installed to the microcontroller board. Both modules provided the function as transmitter or receiver, which means if one of the modules acts as the transmitter, then another one will become the receiver to receive the data sent by the module, so it is a duplex transmission. The communication between both modules is serial data communication, since the communication is between PIC microcontroller and PC, so the asynchronous method is used, which means the data transfer one byte at a time. Besides that, several registers are important in serial port programming, which are the SPBGR (serial port baud rate generator), TXREG (transfer register), RCREG (receive register), TXSTA (transmit status and control register), RCSTA (receive status and control register), and PIR 1 (peripheral interrupt request register 1).[?]

In order to make both wireless modules to recognize each other, the baud rate for both modules must be the same, where the baud rate is 9600 in this project. The baud rate also defined as the data transfer rate between two modules, and the baud rate in PIC microcontroller is programmable. This is done with the help of the SPBRG register. [?] The value to be load into the SPBRG register is depended to the crystal frequency, F_{osc}, the value loaded into SPBRG can be calculated using the following equations (McKinlay et al., 2007):

$$\text{Desired Baud Rate} = F_{osc}/(64X + 64) = F_{osc}/64\,(X+1) \tag{3.12}$$

Where X is the value loaded into the SPBRG register. Since the crystal frequency, F_{osc} = 20 MHz is used in this project, the equation (3.11) will become:

$$\text{Desired Baud Rate} = 312500\text{ Hz}/(X + 1) \tag{3.13}$$

With the desired baud rate equals to 9600, then the value X is equals to:

$$X = (312500 / 9600) - 1 \tag{3.14}$$

$$X = 31.55\text{ (take as 32)} \tag{3.15}$$

Thus, the value to be loaded in the SPBRG register will be 32 in decimal or 20h in hexadecimal, with the BRGH = 0 (the register for high baud rate setting).

TXREG is another 8-bit register used for serial communication in the PIC microcontroller. For a byte of data to be transferred via the TX pin, it must be placed in the TXREG register. The moment a byte is written into TXREG, it is fetched into a register called TSR (transmit shift register). The TSR frames the 8-bit data with the start and stop bits and the 10-bit data is transferred serially via the TX pin. It is similar for RXREG, when the bits are received serially via the RX pin, the PIC microcontroller deframes them by eliminating the stop and start bits, making a byte out of the data received, and then placing it in the RCREG register. (McKinlay et al., 2007)

The TXSTA register is an 8-bit register used to select the synchronous/ asynchronous modes and data framing size, among other things. (McKinlay et al., 2007) In this project, the TXSTA register is loaded with the binary value equals to 0b00100000, to select the 8bit transmission, enabled the transmit, asynchronous mode, and low baud rate. For RCSTA register, it is an 8-bit register used to enable the serial port receive data, among other things. A binary value of 0b10010000 is loaded to RCSTA register in order to enable the serial port, select 8-bit reception, and enable continuous receive.

The TXIF and RCIF bits that locate in the PIR1 register are the important bits in serial communication programming, because both are the interrupt flags for the transmit pin and receive pin. For TXIF flag bit, it will rise when there is a data inside the TXREG register, and indicate the transmit pin is ready to transmit another data when the flag bit goes low. For RCIF flag bit, it indicate the received data is stored in the RCREG, however if fail to copy data in RCREG into a safe place, the data may be loss due to receiving the next data, so checking RCIF flag bit is important to acquire data.

The flow for the serial communication between microcontroller and user (computer) had shown in figure 3.2 and the programming code can refer to Appendix A.

3.7 Snake movement programming (with assist of timers)

As stated at section 3.4, the snake locomotion for this project is serpentine movement. Thus, in order to program the microcontroller to move the servo motors in serpentine curve, the equations (3.3), (3.4), (3.5), and (3.6) are used in the snake movement programming. In order to provide the serpentine curve by the servo motors, the particular duty cycle PWM signals are required to send to the servo motors in every 20ms. Thus, the method of using the timer interrupt is applied to this project, and Timer 1 is used for this method. Besides the Timer 1, the Timer 0 is used in this project as well, the purpose of using Timer 0 is to help the snake like robot in differentiate the obstacle and make the decision to overcome it.

3.7.1 Timer 1 programming

Since the servo motors get the pulse at every 20ms, so Timer 1 is program to count the time and interrupt the microcontroller in every 20ms. Timer 1 is a 16-bit timer, and it consist of two bytes, which referred to as TMR1L (Timer 1 low byte) and TMR1H (Timer1 high byte). (McKinlay et al., 2007) T1CON register is introduced to select the option for Timer 1. In this project, a binary value of 0b00010000 is loaded into T1CON register to set Timer 1 to have 1:2 prescale, and using internal clock source (crystal frequency). After setting up the Timer 1, the initial count values are required for TMR1L and TMR1H. In order to have a count time equals to 20ms, the values to load into both TMR1L and TMR1H registers are calculated as below with crystal frequency equals to 20MHz, and 1:2 prescale:

$$\text{Timer frequency} = 20\text{MHz} /4 /2$$

$$= 2.5\text{MHz} \tag{3.16}$$

$$\text{Step time} = (1/2.5\text{MHz})$$

$$= 0.4\mu\text{s} \tag{3.17}$$

$$x = 20\text{ms}/ 0.4\mu\text{s}$$

$$= 50000 \tag{3.18}$$

$$\text{Decimal value for timer} = 65536 - x$$

$$= 15536\text{d or 3CB0h} \tag{3.19}$$

Because both registers TMR1H and TMR1L are 8-bits register and the calculated value at above is a 16-bits number, so the hexadecimal value above must divided into two bytes, in which the higher byte (3Ch) loaded to TMR1H and lower byte (B0h) loaded to TMR1L. After both registers are loaded, then the TMR1ON bit in T1CON register will set high in order to start the count. Once the Timer 1 is overflowed, the interrupt flag bits, TMRIF will rise, and direct the microcontroller to process the timer's ISR for snake movement as stated in section 3.5.2.

Besides of interrupt function, Timer 1 also used to generate a particular duty cycle PWM signal for servo positioning. For example, in order to position the servo motor to 90 degree (centre), a 1.5ms signal is required, where this can generate by turn on the servo output and turn it off after 1.5ms, the microcontroller is programmed to monitor the value in Timer 1, when the timer had count until 1.5ms, the microcontroller will turn off the servo output. Below is the programming code for microcontroller to determine the time for the pulse width of PWM signal:

```
while (time<T_17point75)            // quit after 2.25ms
 {
        if (time>int_time+ang1)     // compare the value in Timer 1 with the
                                    // required pulse width.
                servo1=0;           // turn off servo if pulse width is
                                    // reached.
        if (time>int_time+ang2)
                servo2=0;
        if (time>int_time+ang3)
                servo3=0;
        get_time();                 // get the value in Timer 1
 }
```

3.8 Sensor programming

Sensor is important for automation of a robot, in order to let the snake like robot has the ability to make decision itself, a sensor is installed on it. Currently, there is only one sensor is used in this project, which is the ultrasonic sensor: MaxSonar- EZ1 sonar range finder, it can sense the object from 0 inch to 254 inches, with two sensing resolution, which is 1 inch resolution at range from 6 inch to 254 inches and 6 inches resolution from 0 inch to 6 inch. Besides that, the interface output format for this sensor can be in pulse width output, analog voltage output, and serial digital output (serial communication). However, the only output format used in this project is analog voltage output, in which the output voltage is range from 0 volt to 2.55

volts with the scaling factor of 10mV per inches. (Maxbotix) In this project, the snake like robot is required to make decision when it senses an object or obstacle within the range of 35cm from the robot, which is the optimize range to let the robot to respond. Since the output format for the sensor is in analog voltage form, and the PIC microcontroller will not respond well to the variation of the analog voltage input, so this project will take concern at the ADC programming as well. The ADC programming will explain in section 3.8.1 and calculation of digital value for desired range will show in section 3.8.2. Where the flow for the snake like robot to respond to the sensor is show in figure 3.3 and the programming code can refer to Appendix A.

3.8.1 ADC programming

In the ADC programming, some of the major characteristics of the ADC are as follows:

- <u>Resolution</u>

  The ADC has *n*-bit resolution, where *n* can be 8, 10, 12, 16, or even 24 bits. The higher-resolution ADC provides a smaller step size, where step size is the smallest change that can be discerned by an ADC. (McKinlay et al., 2007) However, the ADC in PIC18F microcontroller is 10-bit ADC, so the number of steps is equal to 1024.
- <u>Conversion time</u>

  In addition to resolution, conversion time is another major factor in judging an ADC. Conversion time is defined as the time it takes the ADC to convert the analog input to a digital (binary) number. For the ADC in PIC18F microcontroller, the conversion time is dictated by the internal clock source (crystal frequency). (McKinlay et al., 2007)
- <u>Reference voltage</u>

  Reference voltage is one of the important factors that determine the step size of the ADC output. Since the ADC in PIC18F microcontroller is 10-bit resolution, so the step size of the output can calculate if the Vdd of PIC18 is used as the reference voltage (5V). The step size for ADC with 10-bit resolution and used 5V as reference voltage is equals to:

$$\text{Step size} = 5/1024$$

$$= 4.88\text{mV} \quad (3.20)$$

For PIC18F4520 microcontroller, the ADC is control by three registers, which are the ADCON0, ADCON1, and ADCON2. The ADCON0 is the register to choose which analog channels to use, to on the ADC, and start the conversion. ADCON1 is used to select the reference voltage source, and set the pin from AN0 to AN12 either as digital input port or analog input port. ADCON2 is used to set the acquisition time and the conversion clock. Thus, in this project the values '0b00010001', '0b00001010', and '0b10101010' are loaded to the ADCON0, ADCON1, and ADCON2 respectively in order to let ADC to service the data at channel 4 (AN4), turn on the ADC, set the reference voltage as Vdd (5V), set AN4 as the analog input port, set the acquisition time to 12 TAD, and the conversion clock as Fosc/ 32. Besides the ADCON0 value stated above, another binary number '0b00010000' for ADCON0 register is defined in the programming code as well in order to switch off the ADC and so turned off the sensor, and both binary numbers for ADCON0 is defined in programming language as below:

```
#define adc_on 0b00010001           \\ channel to turn on the sensor
#define adc_off 0b00010000          \\ channel to turn off the sensor
```

Figure 3.2 show the method of switching the channel with respect to the input from user, where '1' is to turn on the ADC, and '2' is to turn off the ADC.

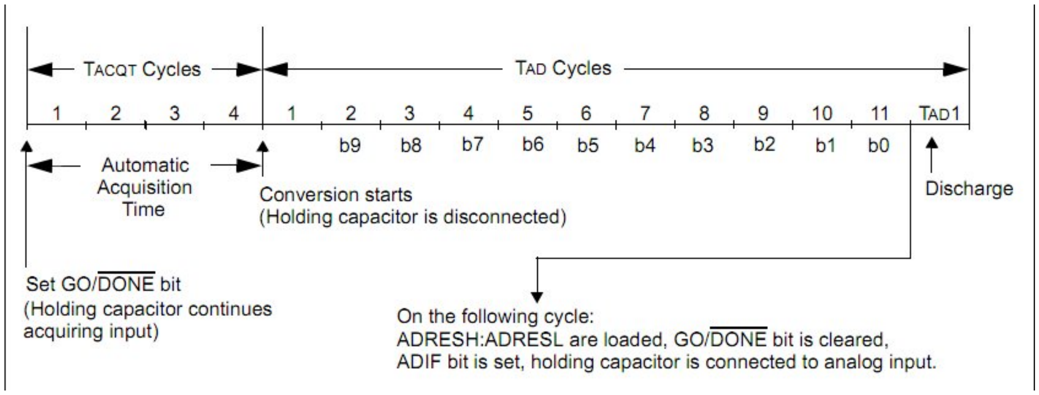

TACQT Cycles
TAD Cycles
1 2 3 4 1 2 3 4 5 6 7 8 9 10 11 TAD1
b9 b8 b7 b6 b5 b4 b3 b2 b1 b0
Automatic Acquisition Time
Conversion starts
(Holding capacitor is disconnected)
Discharge
Set GO/DONE bit
(Holding capacitor continues acquiring input)
On the following cycle:
ADRESH:ADRESL are loaded, GO/DONE bit is cleared,
ADIF bit is set, holding capacitor is connected to analog input.

Figure 3.5: The data acquisition and conversion cycle. (Microchip, 2008)

Figure above show the data acquisition and conversion cycle for ADC, where the T_{AD} is defined as the conversion time per bit and T_{ACQT} is defined as the acquisition time. The T_{ACQT} show above is equals to 4 T_{AD}, and the conversion time for a byte is equals to 12 T_{AD} as well. However, for this project, the acquisition time is set to 12 T_{AD} to make sure all the data can acquire by the ADC. The T_{AD} can be calculated by knowing the conversion clock, so for conversion clock as Fosc/ 32, the T_{AD} is equals to:

$$T_{AD} = 1/ (Fosc/ 32)$$

$$= 1.6\mu s \qquad (3.21)$$

$$\text{Conversion time for 1 byte} = 12\ T_{AD}$$

$$= 19.2\mu s \qquad (3.22)$$

Thus, the minimum conversion time of one byte data is set as 19.2µs, because the 1.6µs is minimum time for T_{AD}. By using the polling method on GO bit in ADCON0, the done of conversion process can be known when the GO bit turn low. After that, the value in ADRESL and ADRESH registers will transfer to a 16-bit variable (result) as shown in the programming code below:

```
ADCON0 = config;          //config will be adc_on or adc_off
delay(10000);             //delay after changing configuration
ADGO = 1;                 //ADGO is the bit 2 of the ADCON0 register
while(ADGO==1);           //ADC start, ADGO=0 after finish ADC progress
result=ADRESH;            //load the value inside ADRESH to result
result=result<<8;         //shift to left for 8 bit
result=result|ADRESL;     //10 bit result from ADC
ADON = 0;                 //adc module is shut off
```

The programming code above also shows the method to read the value in both ADRESL and ADRESH 8-bit register to a 16-bit variable (result) by applied the left switch (<<) and OR(|) operator. The result will then used by the microcontroller to determine the distance of the obstacle.

3.8.2 Calculation of the digital value (converted from analog) for the desired range.

The desired distance in this project is set to 35cm, so that the snake like robot can respond to the obstacle and avoid it. Since the resolution of the sensor used in this project is one inch, so the desired distance will convert to inch first:

$$1\text{cm} = 0.3937 \text{ inch} \tag{3.23}$$

$$\begin{aligned}\text{Desired distance (inch)} &= 35 \times 0.3937 \\ &= 13.78 \text{ inch}\end{aligned} \tag{3.24}$$

With the desired distance in term of inch, the output voltage of the sensor can be calculated:

$$\begin{aligned}\text{Output voltage} &= \text{desired distance (inch)} \times \text{step size (volt per inch)} \\ &= 13.78 \text{ inch} \times 10\text{mV/ inch} \\ &= 138 \text{ mV or } 0.138 \text{ V}\end{aligned} \tag{3.25}$$

With the output voltage from sensor (3.25) and step size (3.19) of the ADC, the digital value can be calculated:

$$\begin{aligned}\text{Result (in decimal)} &= 138\text{mV} / 4.88 \text{ mV} \\ &= 28.28 \text{ (take it as 29)}\end{aligned} \tag{3.26}$$

Thus, the result that gets from ADC will then compare to the decimal value in (3.24) to help the snake like robot to determine whether there have any obstacle within the desired range or not, if got, the snake will try to avoid it.

3.9 Method of snake-like robot used to avoid obstacle

There are two methods for snake like robot to avoid obstacle proposed in this project, where the first method is to stop and centre the snake like robot, then sense the obstacle form, where the second method is to sense the way without obstacle, then move to the particular direction.

3.9.1 First method for obstacle avoidance

For the 1st method, the snake-like robot will stop and centre all the segments once the sensor detects any obstacle within the desired range. After a certain delay, the head of the snake-like robot (1st segment) will turn to end of left (45 degree to the left). Then, the 1st segment will move step by step with a certain step size of angle, until it reaches the end of right. At the same time, the microcontroller will remember the angle once the sensor detects the obstacle, and then accumulate the steps that the obstacle still within the range (obstacle form) to calculate the size of obstacle, and determine the end of the obstacle, then the snake-like robot will determine the direction to move. If there is no outside path for snake-like robot, then the robot will go to the climbing stage to overcome the obstacle (with the height of the obstacle is reachable by the robot).

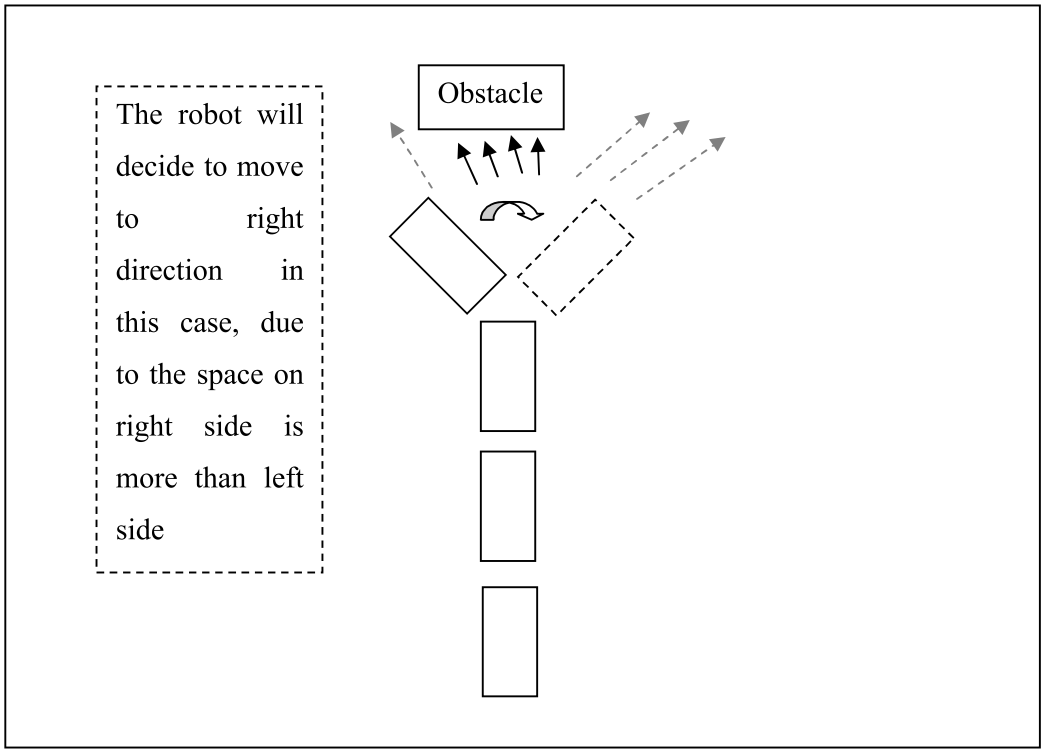

The robot will decide to move to right direction in this case, due to the space on right side is more than left side
Obstacle

Figure 3.6: First method of obstacle avoidance.

3.9.2 Second method for obstacle avoidance

The 2nd method for obstacle avoidance is less time consuming, because the snake-like robot will not stop and have delay as 1st method do, the snake-like robot will sense the solution path while it is moving. Once there is a space after an obstacle, then the snake-like robot will turn to the direction. Below is the programming code for the snake-like robot to decide which direction to move, where the increase and decrease of the value for variable ‘c’ is to change the direction to either left or right as stated in section 3.4:

```
if (result<29)                          // got obstacle within the range
{
        temp_ang=ang1;                  // save the current angle for servo1

        TMR0H=0;                        //Timer 0 = 0
        TMR0L=0;
        T0CONbits.TMR0ON=1;             // on Timer 0
        while (result<29)                   // check whether have solution path
        {                                   // quit the loop after space is found
                if (INTCONbits.TMR0IF==1)           // Timer 0 overflowed
                        break;              // no solution path is found, quit
                else
                        read_adc(channel);  // get the new result
        }
        read_adc(channel);                  //get the current result from sensor
        T0CONbits.TMR0ON=0;                 // off the timer
        if (result>29)                      // if there have solution path
        {
                if (ang1<temp_ang)          // if the angle for servo1 is less than the
                        c+=1000;            // saved angle, then turn left
                else if (ang1>temp_ang)     // if the angle for servo1 is more than
                        c-=1000;            // the saved angle, then turn right
                else
                {
                        if (ang1<3950)
                                c-=1000;
                        else
                                c+=1000;
                }
        }
        else                                // if there have no solution path
                stop = 1;                   // then stop and wait for user command
}
```

The Timer 0 is used in the 2nd method as shown in the programming code above to determine whether there have any solution path within a certain time, if don't have, then the program will lead to stop instruction. The Timer 0 programming will discuss on below.

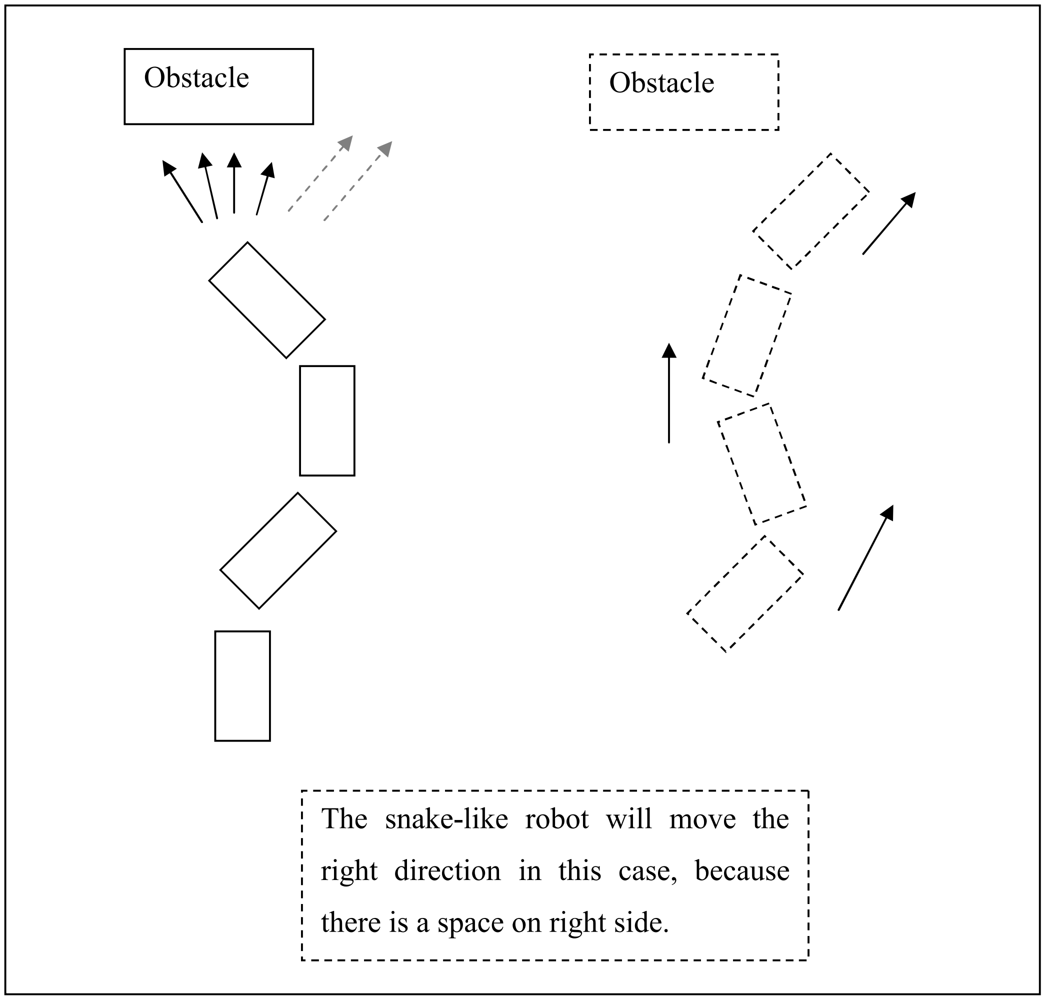

Obstacle
Obstacle
The snake-like robot will move the right direction in this case, because there is a space on right side.

Figure 3.7: Second method of obstacle avoidance.

3.9.3 Timer 0 programming

Since the Timer 0 is just used to helps the snake like robot to make decision, so the interrupt for Timer 0 is disable. The polling method is used for Timer 0, the microcontroller will keep monitor the interrupt flag bit of Timer 0, and break the loop if Timer 0 overflowed before the snake-like robot can sense the solution way to avoid the obstacle. Same as the Timer 1 programming in section 3.7.1, Timer 0 also has the TMR0L and TMR0H registers, and is control by the T0CON register. A binary value ‘0b00000111’ is loaded into the T0CON register to configure the Timer 0 as a 16-bit timer, using internal clock source, and with a prescale of 1:256. Since the Timer 0 is start the count from 0, so the count time is equals to:

$$\text{Timer frequency} = 20\text{MHz} /4 /256$$

$$= 19.53 \text{ kHz} \qquad (3.27)$$

$$\text{Step time} = (1/19.53 \text{ kHz})$$

$$= 51.2\mu\text{s} \qquad (3.28)$$

$$\text{Count time} = 65536 \text{ x } 51.2\mu\text{s}$$

$$= 3.36 \text{ s} \qquad (3.29)$$

This means, if the snake-like robot cannot find the solution path in 3.36 second, then the snake-like robot will stop and centre all segments, and wait for the command from user.

3.9.4 Comparison of both methods

Table 3.2: Comparison of both methods for obstacle avoidance.

	1st Method	2nd Method
Time consuming	Take long time to make decision.	Less time consuming
Accuracy (of making the right decision)	Better accuracy	Good accuracy
Simplicity	Complex	Simple

From the table above, the 2nd method is chosen due to it less time consuming, good accuracy (although not better as 1st method), and simple in programming.

3.10 Method of snake-like robot used in climbing.

In order to overcome the obstacle, climbing ability is required for the snake-like robot. There have two methods of climbing process proposed to this project, which are the automatically climbing process, and manually control climbing process.

3.10.1 Automatically climbing process (1st method)

For 1st method, the snake-like robot is able to climb over the obstacle by itself. Once the program is enter to the climbing process, the snake-like robot will in the situation that all segments are centred, and then the 1st and 2nd segments of the robot will raise in the form shown in figure below to sense the height for the obstacle:

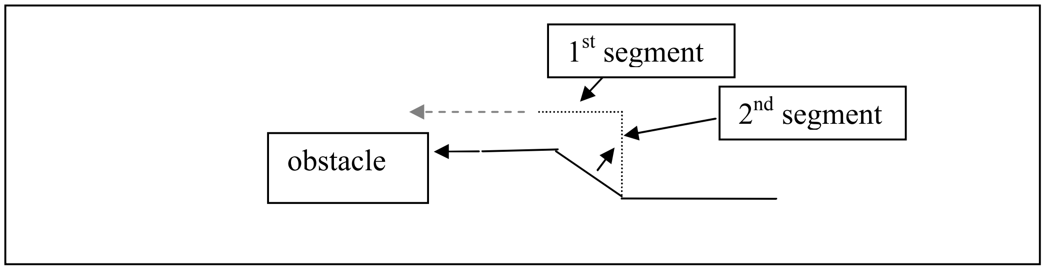


Figure 3.8: The method to sense the height of obstacle.

After get the height of the obstacle, the snake-like robot will start the climbing process as shown in the figure below:

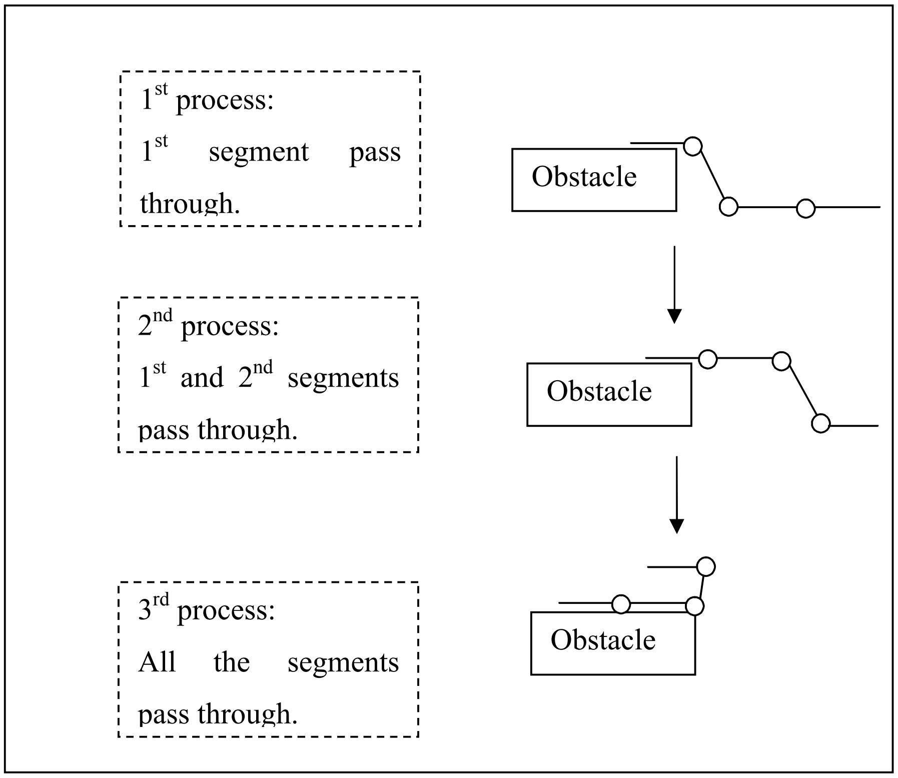

1st process:
1st segment pass through.
2nd process:
1st and 2nd segments pass through.
3rd process:
All the segments pass through.
Obstacle
Obstacle
Obstacle

Figure 3.9: The automatically climbing process.

From the figure above, the snake-like robot will raise the 1st and 2nd segment in the 1st process, and power the motor to let the snake-like robot to move forward, and so the 1st segment will pass through the obstacle, then the 3rd segment will raise and power the motor again to let 2nd segment to pass through the obstacle, for the 3rd and 4th segment, both will raise together, and the 1st and 2nd segment will move forward so that all the segments will pass through the obstacle.

3.10.2 Manual climbing process (2nd method)

In this method, the snake-like robot will climb over the obstacle with the assist of user command, the user will send the command to raise or lower each segment step by step. Below is part of the programming code for the snake-like robot to respond to user command in climbing method:

```
case 'y':                                  // to raise 1st segment
        if (stop ==1)
        center_ang5+=400;
        break;
case 'h':                           // to lower 1st segment
        if (stop ==1)
        center_ang5-=400;
        break;
case 'u':                           // to raise 2nd  segment
        if (stop ==1)
        center_ang6+=400;
        break;
case 'j':                           // to lower 2nd  segment
        if (stop ==1)
        center_ang6-=400;
        break;
case 'i':                           // to raise 3rd  segment
        if (stop ==1)
        center_ang7+=400;
        break;
case 'k':                           // to lower 3rd  segment
        if (stop ==1)
        center_ang7-=400;
        break;
```

The reasons of using manual climbing process are the flexibility of the climbing process, ease to determine the form of climbing, and stability of the snake-like robot can be control.

3.10.3 Comparison of both climbing method

Table 3.3: Comparison of both methods for obstacle avoidance.

	Automatic climbing	Manual climbing
Flexibility	Fixed	Can be change to adapt in different environment

Stability	Hard to control since the weight of snake-like robot is not uniform.	Can be control by the user in command the climbing process.

The table above show the comparison between the automatic climbing method and manual climbing method. From the table, the manual climbing seems to have more advantageous in flexibility and stability, this also the reason manual climbing is used in this project. However, this may limit the ability of snake-like robot when it is in the closed environment without the monitor of the user.

3.11 DC motor programming

DC motor is used to provide the driving force for the snake-like robot in this project. Although the snake-like robot can generate the driving force by itself with snake-like locomotion, but a DC motor is used to improve the speed of the snake-like robot. Thus, the DC motor programming is required to order the motor to stop or move. Since the DC motor that used in this project is not a high speed motor, so this section will not take consideration on the variation of speed, which means the duty cycle used to power the DC motor is either 0% or 100%.

In the DC motor programming, the CCP (Compare Capture Pulse-Width-Modulation) of PIC18F4520 is used to generate the PWM of certain duty cycle to control the motor. There are several registers that must take consideration in PWM programming using CCP, which are the PR2 register, T2CON register, CCP1CON register, and CCPR1L register. PR2 register is used to set the period of the PWM, where T2CON is used to configure and ON Timer 2, CCP1CON is used to select the mode to PWM and CCPR1L is some percentage of PR2 register, because the value loaded into this register is to determine the percentage of duty cycle.

In this project, the value for CCPR1L may either be 0 or the same as value in PR2 register in order to generate 0% or 100% duty cycle, as the PR2 in this project is set to 255, so the CCPR1L will have a value of 255. The Timer 2 is set to no prescaler and postscaler, where the use of Timer 2 is to count the period for value in PR2, and the Timer 2 will reset once the value is same as PR2.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Results

There are two important factors to examine in design the snake-like robot, which are the serpentine curve of the snake-like robot, and the accuracy of the ultrasonic sensor. As stated at section 3.4, the serpentine curve is depend on the three variables, *a*, *b*, and *c*. Thus, the analysis on the waveform of each joint with different values of the three variables is required.

4.1.1 The waveforms of each joint. (a=0.7854, b=3π, c=0)

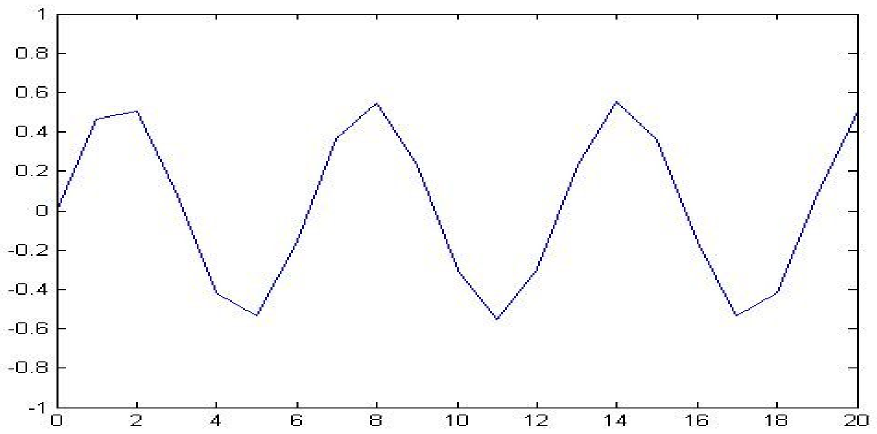


Figure 4.1: The waveform for 1st joint.

Figure above is the waveform for 1st joint, where the variables: a=0.7854rad (45degree), b=3π, c=0. The waveform above shows that the maximum amplitude is approximately to 0.58 rad, the time for each cycle is around 6.3t, and start at 0degree when t=0. The 0.58 rad equals to:

$$(0.58/\pi)*180 = 33.23 \text{ degree} \tag{4.1}$$

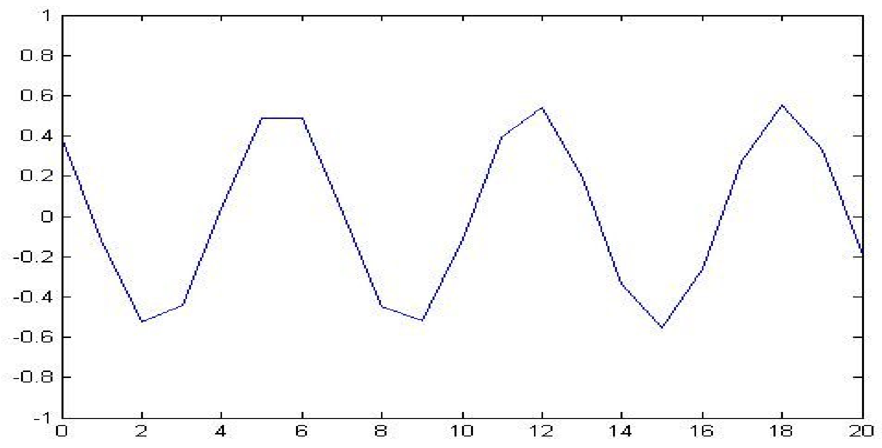


Figure 4.2: The waveform for 2nd joint.

Figure above is the waveform for 2nd joint, where the variables: a=0.7854rad (45degree), b=3π, c=0. Where the amplitude at t=0 is calculated below:

$$(0.39/\pi)*180 = 22.35 \text{ degree} \tag{4.2}$$

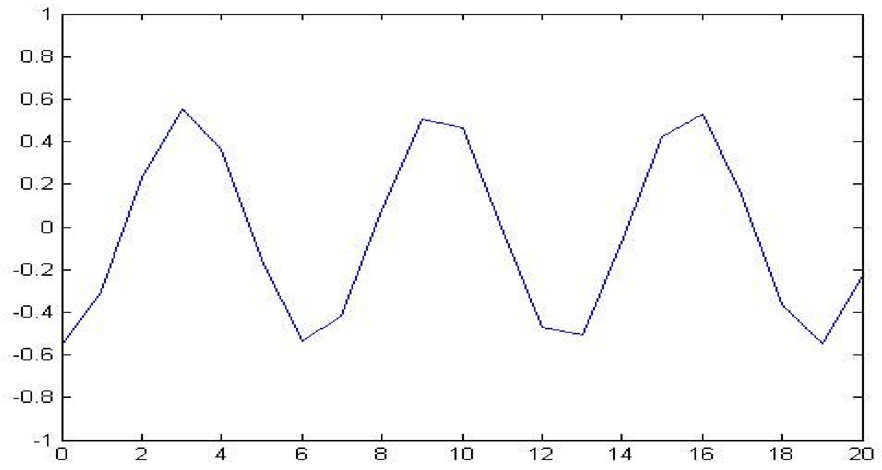


Figure 4.3: The waveform for 3rd joint.

Figure above is the waveform for 3rd joint, where the variables: a=0.7854rad (45degree), b=3π, c=0.

4.1.2 The waveforms of 1st joint with different value of *a*.

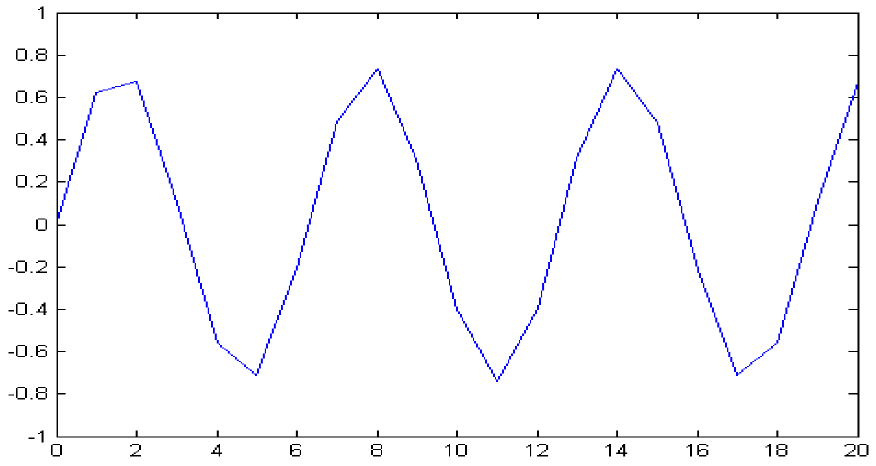


Figure 4.4: The waveform for *a*=1.047

Figure above are the waveform of 1st joint, where the variables: a=1.047rad (60degree), b=3π, c=0.

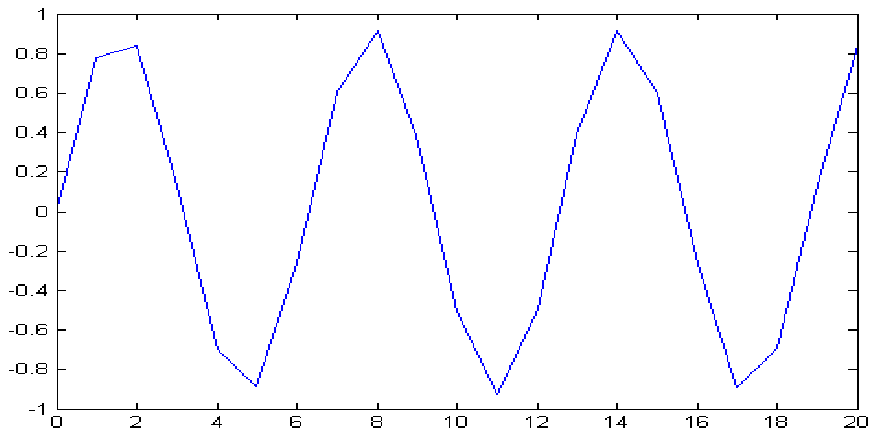


Figure 4.5: The waveform for *a*=1.309

Figure above are the waveform of 1st joint, where the variables: a=1.309rad (75degree), b=3π, c=0.

4.1.3 The waveforms of 1st joint with different value of *c*.

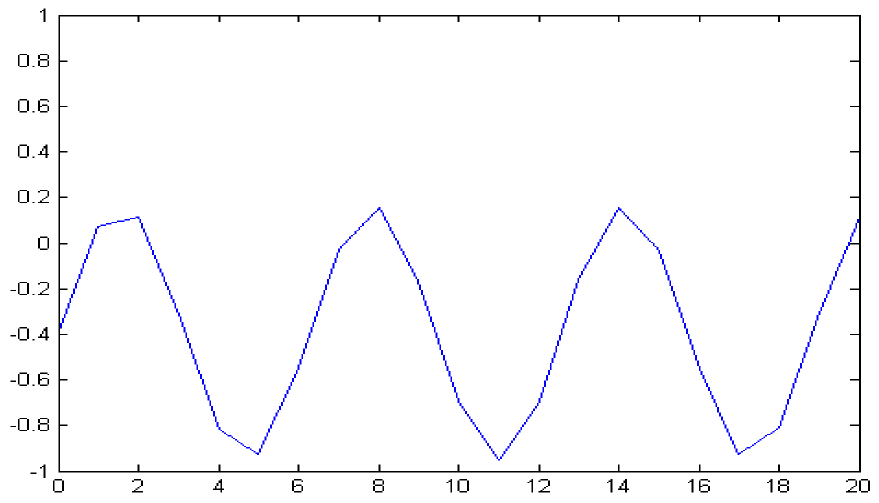


Figure 4.6: The waveform for *c=π/2*

Figure above are the waveform of 1st joint, where the variables: a=0.7854rad (45degree), b=3π, c=π/2.

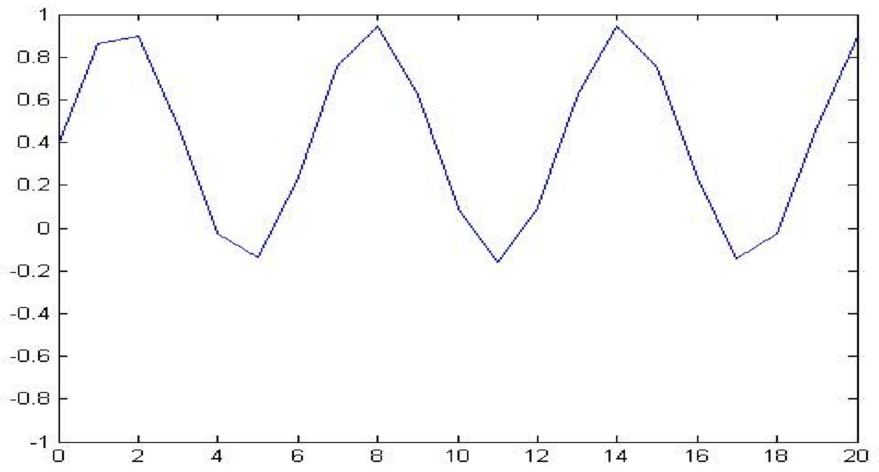


Figure 4.7: The waveform for *c=-π/2*

Figure above are the waveform of 1st joint, where the variables: a=0.7854rad (45degree), b=3π, c=-π/2.

4.1.4 The results of ultrasonic sensor in different distance (output voltage).

Table 4.1: Results of ultrasonic sensor in different distance

Output Voltage / Distances	1st result (mV)	2nd result (mV)	3rd result (mV)	Average (mV)
4inch	64.5	64.8	64.7	64.67
8inch	75.1	74.2	73.9	74.40
12inch	112.8	112.7	112.6	112.70
16inch	151.0	150.8	151.2	151.00

The output voltage of each distance can be calculated by using equation (3.25):

$$\text{Output voltage (4 inch)} = 4 \text{ inch x } 10\text{mV/ inch} = 40 \text{ mV} \quad (4.3)$$

$$\text{Output voltage (8 inch)} = 8 \text{ inch x } 10\text{mV/ inch} = 80 \text{ mV} \quad (4.4)$$

$$\text{Output voltage (12 inch)} = 12 \text{ inch x } 10\text{mV/ inch} = 120 \text{ mV} \quad (4.5)$$

$$\text{Output voltage (16 inch)} = 16 \text{ inch x } 10\text{mV/ inch} = 160 \text{ mV} \quad (4.6)$$

The % error is calculated as well at below:

% error(4 inch) = |(64.67 – 40)/ 40| * 100%

= 61.68% (4.7)

% error(8 inch) = |(74.4 – 80)/ 80| * 100%

= 7% (4.8)

% error(12 inch) = |(112.7 – 120)/ 120| * 100%

= 6.08% (4.9)

% error(16 inch) = |(151 – 160)/ 160| * 100%

= 5.63%

(4.10)

Table 4.2: Comparison between measured value and calculated value

Distances (inch)	Measured Output Voltage (mV)	Calculated Output Voltage (mV)	Percentage Error (%)
4	64.67	40	61.68
8	74.40	80	7.00
12	112.70	120	6.08
16	151.00	160	5.63

4.2 Discussion

In the project, the waveforms of each joint are plotted and examined. Figure 4.1 shows the waveform for 1st joint, where the variables: a=0.7854rad (45degree), b=3π, c=0. The waveform above shows that the maximum amplitude is approximately to 0.58 rad, the time for each cycle is around 6.3t, and start at 0degree when t=0.

Figure 4.2 shows the waveform for 2nd joint, where the variables: a=0.7854rad (45degree), b=3π, c=0. Same as 1st joint, the maximum amplitude is about 0.58rad (33.23 degree), and having the cycle time approximately to 6.3t. However, the direction of the 2nd joint is opposite to the 1st joint, and the angle is on 0.39rad when t=0.

Figure 4.3 shows the waveform for 3rd joint, where the variables: a=0.7854rad (45degree), b=3π, c=0. Same as 1st joint, the maximum amplitude is about 0.58rad (33.23 degree), having the cycle time approximately to 6.3t, and same direction as 1st joint but starting at -0.58rad, which equals to minimum amplitude of the waveform.

From the waveforms, we can know that the maximum degree that each joint can reach is about 33.23 degree, however, the maximum degree that the snake-like robot can reach is 45degree, due to the relative angle between each joint. Besides that, the direction for 2nd joint is different with the direction for 1st joint and 3rd joint, so the serpentine curve can be form.

Besides the waveforms of each joint are examined, the waveform for different values of variable is examined as well. For different value of a, the amplitude of each waveform will increase as the value of a is increased, and decrease the amplitude as a is decrease. For different value of c, will shift the waveform either to upward direction or downward direction depend on the sign of value c, shift upward when the c is negative, and shift downward when c is positive.

The accuracy of the ultrasonic sensor is examined. The percentage errors between the measured output voltage and calculated output voltage of the sensor are low except the percentage errors of output voltage for distance at 4inch, this may due to the different step resolution of the sensor. As stated in section 3.8, the ultrasonic sensor used in this project has a 6 inches resolution from 0 inch to 6 inch, so the measured output voltage at 4 inch is actually the output voltage for 6 inch:

$$\text{Output voltage (6 inch)} = 6 \text{ inch} \times 10\text{mV/ inch}$$

$$= 60 \text{ mV} \tag{4.11}$$

$$\text{\% error(6 inch)} = |(64.67 - 60)/ 60| * 100\%$$

$$= 7.78\% \tag{4.12}$$

Thus, the actual percentage error is 7.78%. Therefore, the accuracy of the ultrasonic sensor can consider high due to the small percentage errors of output voltage at different distances.

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

As a conclusion for programming part, the programming of serpentine movement on the snake-like robot is done, the snake-like robot able to make decision to overcome the obstacle, and able to respond to the user command. In generally, all of the objectives stated are achieved in this project:
-The snake-like robot able to provide snake-like locomotion.
-The snake-like robot can make the decisions itself when facing problems (obstacles).
-The distance can be detected with the assist of sensors.
-The snake-like robot can lift at a certain height so that it can climb stair or climb over the obstacles.
-The snake-like robot can remotely control by the user.

5.2 Recommendation and future development

Firstly, a camera is recommended to install to the snake-like robot in order to let the snake-like robot have high level of decision making instead of just using the ultrasonic sensor, because the surface form of the obstacles is impossible to get by just using the ultrasonic sensor. With a camera, the user can monitor the movement of the snake-like robot even in the closed environment where human cannot be inside. Besides that, the image processing technology like machine vision can program to the snake-like robot, so that the robot can make decision based on the processed image.

Secondly, more sensors could be added to the snake-like robot, to allow it to know its position and the environment that surround it. The gyroscope can be added, so that the snake-like robot can stabilize itself in the climbing process.

Lastly, the additional features such as sense for fire, detect human breathing, and other useful features can be added to the snake-like robot, so that it can help human to perform rescue services in the environment that human can’t reach.

REFERENCES

Junyao Gao; Xueshan Gao; Wei Zhu; Jianguo Zhu; Boyu Wei; , "Design and research of a new structure rescue snake robot with all body drive system," *Mechatronics and Automation, 2008. ICMA 2008. IEEE International Conference on* , vol., no., pp.119-124, 5-8 Aug. 2008

Hua Liu; Guozheng Yan; Guoqing Ding; , "Research on the locomotion mechanism of snake-like robot," *Micromechatronics and Human Science, 2001. MHS 2001. Proceedings of 2001 International Symposium on* , vol., no., pp.183-188, 2001

McKenna, J.C.; Anhalt, D.J.; Bronson, F.M.; Brown, H.B.; Schwerin, M.; Shammas, E.; Choset, H.; , "Toroidal skin drive for snake robot locomotion," *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on* , vol., no., pp.1150-1155, 19-23 May 2008

http://mcu-programming.blogspot.com/2006/09/servo-motor-control.htm

Muhammad Ali Mazidi, Rolin McKinlay, Danny Causey (2007). *PIC Microcontroller and Embedded Systems: Using Assembly and C for PIC18* (2th ed.). DeVry University: Prentice Hall.

Maxbotix. (2005). Retrieved Jan 2005, from MaxSonar EZ-1 Data Sheet: http://www.cytron.com.my/datasheet/sensor/LVEZ1.pdf

Kai-Hsiang Chang; Yung-Yaw Chen; , " Efficiency on Snake Robot Locomotion with Constant and Variable Bending Angles" *IEEE International Conference on Advanced Robotics and its Social Impacts*, vol., no., pp., 23-25 Aug. 2008

APPENDICES

APPENDIX A: C Programming Code

```
#include <P18f4520.h>
#include "delays.h"
#include "usart.h"
#include "math.h"
#include "stdlib.h"
#include "stdio.h"

#pragma config OSC = HS

#define pi 3.142
#define A_default 1000
#define B_default 3*pi
#define C_default 0
#define seg 4
#define led PORTDbits.RD1
#define sw1 PORTBbits.RB0
#define servo1 PORTBbits.RB1
#define servo2 PORTBbits.RB2
#define servo3 PORTBbits.RB3
#define servo4 PORTBbits.RB4
#define servo5 PORTBbits.RB5
#define servo8 PORTBbits.RB7
#define servo7 PORTBbits.RB6
#define servo6 PORTDbits.RD7
#define ADGO ADCON0bits.GO
#define ADON ADCON0bits.ADON
#define adc_on 0b00010001
#define adc_off 0b00010000
#define motor_spd CCPR1L


volatile float a=A_default;
volatile float b=B_default;
volatile float c=C_default;

volatile float alpha;
volatile float beta;
volatile float gamma;
volatile float t=0;

volatile unsigned int channel;
volatile int center_ang1=3950;
volatile int center_ang2=3950;
volatile int center_ang3=3950;
volatile int center_ang4=3950;
volatile int center_ang5=3950;
volatile int center_ang6=3950;
volatile int center_ang7=3750;
volatile int center_ang8=4150;
volatile int temp_ang;
volatile int ang1,ang2,ang3,ang4,ang5;
volatile int ang6,ang7,ang8,ang9,ang10;
volatile int time;

volatile float z;
volatile float func;
volatile float height;
volatile float speed=0.01*pi;


volatile unsigned int T_17point75;
volatile unsigned int int_time;

unsigned char on;
unsigned char start;
unsigned int distance;
unsigned int result;

void chk_rx_isr(void);
void chk_timer_isr(void);
void T1_isr(void);
void Rx_isr(void);
void get_time(void);
void read_adc(char config);
void delay(unsigned long data);
void stop_isr(void);



#pragma code Rx_int=0x0008                //High-priority interrupt
void Rx_int(void)                         //Rx-pin interrupt
{
    _asm
        GOTO chk_rx_isr
    _endasm
}
```

```
#pragma code Timer_int=0x0018              //Low priority interrupt
void Timer_int(void)                       //Timer interrupt
{
    _asm
        GOTO chk_timer_isr
    _endasm
}

#pragma interruptlow chk_timer_isr
void chk_timer_isr(void)
{
   if (PIR1bits.TMR1IF==1)
{
      if (distance==2000)
          stop_isr();
      else
          T1_isr();
}

}

#pragma interrupt chk_rx_isr
void chk_rx_isr(void)
{
   if(PIR1bits.RCIF==1)
      Rx_isr();
}

void Rx_isr()                               // Rx ISR
{
    int input;
    while(PIR1bits.RCIF==0);               //wait until RCIF become 1
    input=RCREG;                            //read input from RCREG
    switch(input)                          //cases for user's command
    {
       case '1':                           //start the robot with sensor ON
          distance=0;
          channel=adc_on;
          break;
       case '2':                           //start the robot with sensor OFF
          distance=0;
          channel=adc_off;
          break;
       case 's':                           //stop the robot
          distance=2000;
          break;
       case 'd':                          //turn right
          c += 1000;
          gamma=-c/seg;
          break;
```

```
case 'a':                              //turn left
   c -= 1000;
   gamma=-c/seg;
   break;
case 'y':                               //1st segment move upward
   if (distance==2000)
      center_ang6+=400;
   break;
case 'h':                               //1st segment move downward
   if (distance==2000)
      center_ang6-=400;
      break;
case 'u':                                //2nd segment move upward
   if (distance==2000)
      center_ang4+=400;
      center_ang5-=400;
   break;
case 'j':                                //2nd segment move downward
   if (distance==2000)
      center_ang4-=400;
      center_ang5+=400;
   break;
case 'i':                                 //3rd segment move upward
   if (distance==2000)
      center_ang7+=400;
      center_ang8-=400;
   break;
case 'k':                                //3rd segment move downward
   if (distance==2000)
      center_ang7-=400;
      center_ang8+=400;
   break;
case 'Y':                                //1st segment move to right
   if (distance==2000)
      center_ang1+=400;
   break;
case 'H':                                //1st segment move to left
   if (distance==2000)
      center_ang1-=400;
   break;
case 'U':                                //2nd segment move to right
   if (distance==2000)
      center_ang2+=400;
   break;
case 'J':                                //2nd segment move to left
   if (distance==2000)
      center_ang2-=400;
   break;
case 'I':                                 //3rd segment move to right
   if (distance==2000)
```

```
                center_ang3+=400;
            break;
        case 'K':                               //3rd segment move to left
            if (distance==2000)
                center_ang3-=400;
            break;
        case 'R':                                  //reset data to default
            c=0;
            center_ang1=3950;
            center_ang2=3950;
            center_ang3=3950;
            center_ang4=3950;
            center_ang5=3950;
            center_ang6=3950;
            center_ang7=3950;
            center_ang8=3950;
            break;
        case 'r':
            motor_spd=102;                         //move the motor
            break;
        case 'f':
            motor_spd=0;                           //stop the motor
            break;
        default:
            break;
    }
}



void T1_isr()
{
    TMR1H=0x3C;                                //20ms for Timer1
    TMR1L=0xB0;

    servo1=1;                                  //Start the duty cycle for servo motors
    servo2=1;
    servo3=1;

    get_time();                                //take current time
    int_time=time;                             //save to variable 'int_time'
    T_17point75=0x52A9;                        //value=17.75ms

    T1CONbits.TMR1ON=1;                        //On Timer1
    while (time<T_17point75)
    {
        if (time>int_time+ang1)                //if time reach
            servo1=0;                        //end of the duty cycle for servo motors
        if (time>int_time+ang2)
            servo2=0;
        if (time>int_time+ang3)
            servo3=0;

        get_time();                                 //get the current time
    }

    ang1=(alpha*sin(t)+gamma+3950);                 //snake equations
    ang2=(alpha*sin(t+1*beta)+gamma+3950);
    ang3=(alpha*sin(t+2*beta)+gamma+3950);

    t+=speed;
    if (t >= 2*pi){
        t = 0;
    }
    else if (t < 0){
        t = 2*pi;
    }


    PIR1bits.TMR1IF=0;                              //clear the interrupt flag for Timer 1
}

void stop_isr()
{
     TMR1H=0x3C;                                    //2.25ms=5625H
     TMR1L=0xB0;


     servo1=1;

     servo2=1;

     servo3=1;

     servo4=1;

     servo5=1;

     servo6=1;

     servo7=1;

     servo8=1;



     get_time();
     int_time=time;
     T_17point75=0x52A9;
```

```
	T1CONbits.TMR1ON=1;

	while (time<T_17point75)
							{
			if (time>int_time+ang1)
				servo1=0;
			if (time>int_time+ang2)
				servo2=0;
			if (time>int_time+ang3)
				servo3=0;
			if (time>int_time+ang4)
				servo4=0;
			if (time>int_time+ang5)
				servo5=0;
			if (time>int_time+ang6)
				servo6=0;
			if (time>int_time+ang7)
				servo7=0;
			if (time>int_time+ang8)
				servo8=0;
			get_time();
		}



			ang1=center_ang1;		//set all servo motors to center position
			ang2=center_ang2;
			ang3=center_ang3;
			ang4=center_ang4;
			ang5=center_ang5;
			ang6=center_ang6;
			ang7=center_ang7;
			ang8=center_ang8;

		PIR1bits.TMR1IF=0;
		t=0;				//reset the value for t
}


void main(void)
{
		while(1)
		{
			unsigned char n;
			unsigned char Prompt[]="Select Environment Mode:1)Flat surface
						2)Uneven surface";

			distance=2000;			//stop the robot initially
```

```
TRISB=0b00000001;          //set portB as output except RB0
TRISDbits.TRISD7 = 0;      //RD7 as output
TRISDbits.TRISD1 = 0;      //RD1 as output
TRISAbits.TRISA5 = 1;      //RA5 as input
TRISC=0b10000000;          //RC7 as input, and other pin for PortC are output

PR2=255;                   //the period of PWM at 19.53kHz
T2CON=0b00000100;          //start the Timer2 with no prescaler and postscaler
CCP1CON=0b00001100;        //set the CCP to PWM mode
motor_spd=0;               //stop the motor

ADCON1 = 0b00001010;       //reference voltage=Vdd, AN4 as analog input port
ADCON2 = 0b10101010;       //Tacqt=12Tad, conversion clock=Fosc/32

T1CON=0b00010000;          //with prescaler= 1:2
T0CON=0b00000111;          //with prescaler= 1:256
TMR1H=0x3C;
TMR1L=0xB0;                //3CB0H=20ms

SPBRG = 0x20;              //baud rate=9600
TXSTA=0b00100000;          //TXEN=1
RCSTA=0b10010000;          //SPEN=1,CREN=1
PIE1bits.RCIE=1;           //enable the receive interrupt bit
RCONbits.IPEN=1;           //enable for different priority interrupt
IPR1bits.TMR1IP=0;         //Timer 1 as low priority interrupt
INTCONbits.TMR0IF=0;       //clear Timer0's interrupt flag
PIR1bits.TMR1IF=0;         //clear the timer1 interrupt flag
PIE1bits.TMR1IE=1;         //enable the timer1 interrupt flag
INTCONbits.GIEH=1;         //enable all high priority interrupt
INTCONbits.GIEL=1;         //low priority interrupt enable
T1CONbits.TMR1ON=1;        //On Timer1

delay(100000);



          for(n=0;n<55;n++)     //send text
          {
                    while(PIR1bits.TXIF==0);
                    TXREG=Prompt[n];   //loaded data to send
          }

          while(1)
          {
                    if (distance==0)                //on by user input
                              break;
                    if (sw1==0)                     //on by switch
                              distance=0;
                    channel=adc_on;
                              break;
```

```
}

while(1)
{

        gamma=-c/seg;
        beta=b/seg;
        z=(sin(beta));

        if (z<0)
                func=z-(2*z);

        else
                func=z;

        alpha=a*func;


        read_adc(channel);

        if (result<29)
        {
                temp_ang=ang1;

                TMR0H=0;
                TMR0L=0;
                T0CONbits.TMR0ON=1;

                while (result<29)
                {
                        if (INTCONbits.TMR0IF==1)
                                break;

                        else
                                read_adc(channel);
                }
                read_adc(channel);
                T0CONbits.TMR0ON=0;
                if (result>29)
                {
                        if (ang1<temp_ang)

                                c+=1000;

                        else if (ang1>temp_ang)

                                c-=1000;

                        else
                        {
```

```
                                        if (ang1<3950)
                                                c-=1000;
                                        else
                                                c+=1000;
                                }
                        }
                        else

                        {
                                distance=2000;



                        }

                }


        }





    }
}



void get_time()                                         //get value in 16-bit
Timer1
{
    time=TMR1H;
    time=time<<8;                                       //shift to left for 8 bit
    time=time|TMR1L;                                //16-bit result from Timer1
}


void read_adc(char config)
{

    unsigned long result_temp=0;

    ADCON0 = config;
    delay(10000);                   // delay after changing configuration

        ADGO = 1;                   //ADGO is the bit 2 of the ADCON0 register
        while(ADGO==1);//ADC start, ADGO=0 after finish ADC progress
```

```
                result=ADRESH;
                result=result<<8;                  //shift to left for 8 bit
                result=result|ADRESL;              //10 bit result from ADC



        ADON = 0;                                  //adc module is shut off

        if(result<29)
                led=1;
        else
                led=0;


}

void delay(unsigned long data)                     //delay function, the delay time
{                                                  //depend on the given value
        for( ;data>0;data-=1);
}
```