University of Florida

Department of Electrical and Computer Engineering

EEL 4665/5666

Intelligent Machines Design Laboratory

Hex Guard

Student Name: Tianyu Gu

Email: tianyu.gu@ufl.edu

TAs: Ryan Chilton, Josh Weaver

Instructors: Dr. A. Antonio Arroyo, Dr. Eric M. Schwartz

Contents

Abstract

Making land-mines is very cheap, but disarming them is not. Nowadays, there are millions of mines buried in the earth, which is taking lives almost every hour all over the globe. Statistics shows that it will cost 33 billion dollars to disarm all the land-mines in the world, and the disarming activity itself has also taken innumerous lives of soldiers. A terrain robot capable of moving on rugged road could potentially become a solution to this. This 'Hex Guard', which mimics the walking gait of RHex built by Boston Dynamics, patrols around in an area to search for metal objects, which could be eventually be weapons or land-mines when scaled up (Fig.1)

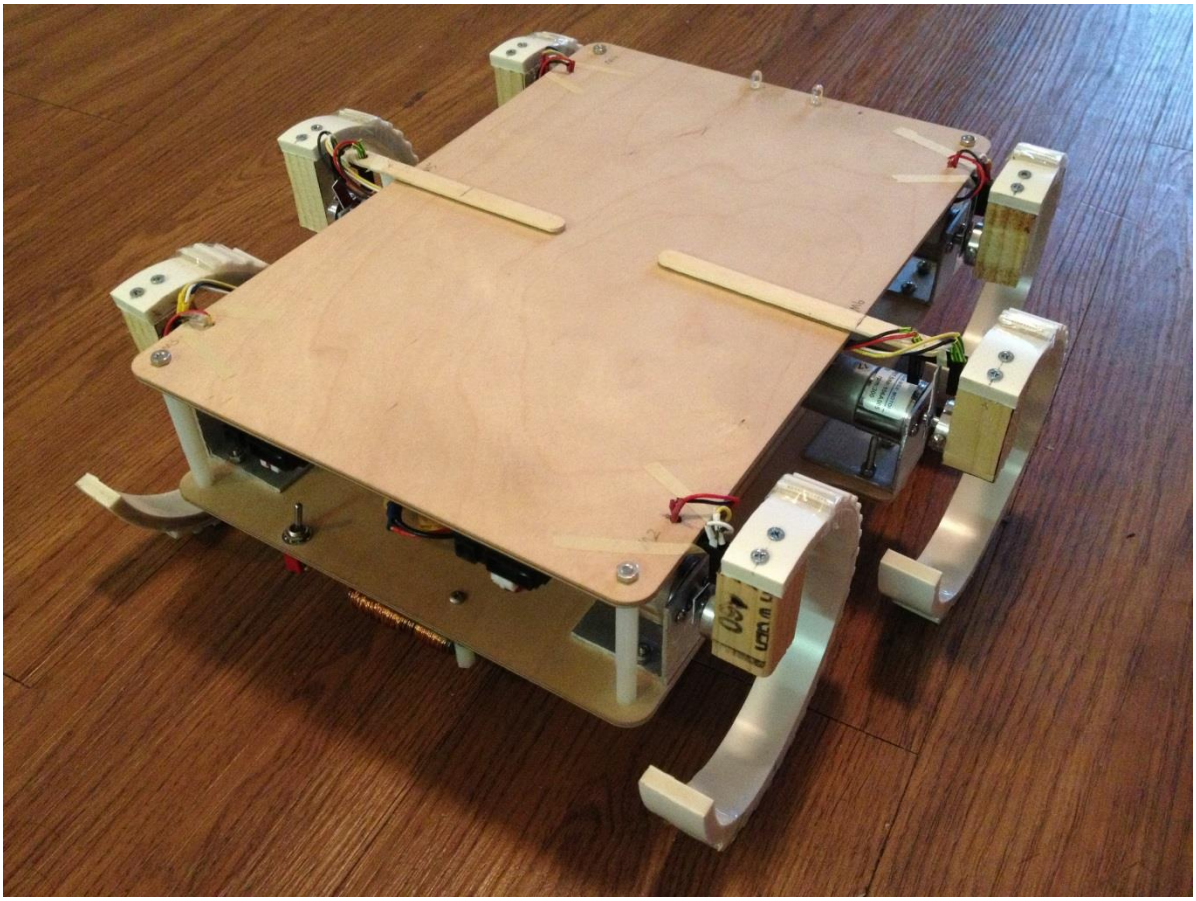

Fig.1 Hex Guard

Introduction

I was first inspired by the six-leg walking robot ‘RHex’ built by Boston Dynamics, I soon developed a fever for its walking gait since it can continuously walk despite obstacles or flipping. Then I start to think of using it for a specific purpose. Nowadays wars are taking place all over the world almost every day. The biggest loss of war is the life of soldiers and civilians. Soldiers typically use a metal detector device to detect weapons or mines. But at the same time that soldier also risks losing his/her life. Hex Guard, however, is less likely to trigger a mine due to its weight, or to be noticed by terrorists. We start with introducing the integrated system of Hex Guard, and continue with its mechanical design, actuation, and sensor system. In the concluding section, prospective features for Hex Guard are mentioned.

Integrated System

Hex Guard uses Epiphany board with ATXmega processor, one 3 cell Lipo battery for actuation, one 9V alkaline battery for board powering, one quad and one dual motor controller, six DC gear motors with photo-interrupters, two IR sensors, an accelerometer, and a metal detector. Epiphany board handles all the data collection, decision making and execution. Motor controllers control the direction and speed of DC motors with PWM signal coming from the board. Photo-interrupters provide feedback signal to control the ‘start/stop’ of motors. It uses two IR sensors to perform basic object avoidance. A metal detector is mounted beneath the robot’s platform to detect metal objects. An accelerometer is also mounted in case the robot flips. Finally, two LEDs on top of the platform indicate different behaviors. Below is a block diagram of the system.

Mobile Platform

The platform of Hex Guard is designed to be symmetric about horizontal plane. The main body is made of two pieces of 3mm thick wood that are connected by plastic standoffs. The bottom of the platform is mainly comprised of Epiphany board, batteries, and actuation components. The top part of the platform mainly holds sensors such as photo-interrupters and IR sensors (Fig2). Top cover is assembled into the main platform with removable connectors. This design makes it easy to access and maintain since the top cover is ready to remove.

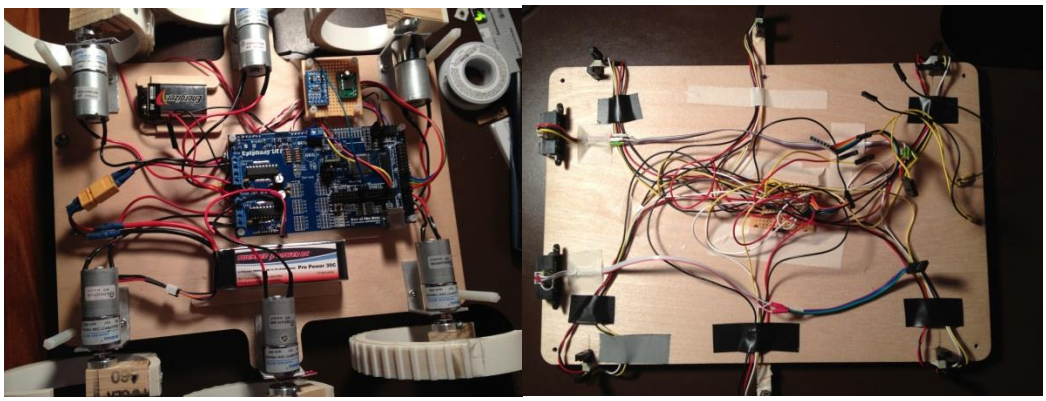

Fig.2 Bottom and Top Platform

Hex guard uses PVC as the material for its leg mainly for two reasons: First, unlike wood, PVC leg can bend a little bit, thus it will absorb shocks when hitting the ground. Secondly, it is pretty easy to cut the leg from a PVC pipe. Each leg is then attached with a piece of mounting tape to give certain friction while at the same time absorb shock.

Attribute	Value
Body Height	110 mm
Overall Width	385 mm
Length	350 mm
Leg to Leg Spacing	130 mm
Ground Clearance	53 mm
Leg Diameter	114 mm
Total Weight	Approx. 4 kg

Tab.1 Physical Properties

Actuation

Hex Guard uses six DC motors to drive its legs. DC gear motor with high torque and moderate gear ratio is preferred for the following considerations. First, since it will perform some heavy-duty tasks such as climbing and standing up, Hex Guard would require high torque output for each of its leg. As we were taught before, the speed of a motor is anti-proportional to its torque. Thus big gear ratio is can achieve powerful torque with low speeds. However, Hex Guard is also supposed to run quickly on flat road with moderate torque, so we don't want to totally give up speed. Such a tradeoff between torque and speed result in our considerations for DC gear motor.

Below chart shows the parameters of motor.

Attribute	Value
Power	8 W
Gear Ratio	33:1
Voltage	12 V
RPM	300
Torque	576.5 mNm (5 Lb Inch)
Shaft Diameter	4 mm

Tab.2 Motor Parameter

Hex Guard will first stand up using all six legs. Basically it has two walking gaits to choose from (details will be introduced in later chapter). One of them is the three-by-three walking gait that is intended for smooth surfaces. In the tripod gait, three of the legs (as shown in Fig.3), which are synchronized with light gates, start to rotate from 0 to 360 degrees. Once the photo-interrupters sense one complete revolution of the first three legs, the other three legs start to rotate while the first three will hold in static position. Repeating these two cycles gives a continuous sequence walking of the robot. Turning is realized by reversing one of the motors in the same group. For

example, if the robot is turning right, then the motors on the left side will rotate forward, while the one on the right side will rotate backward. After repeating this action for several times, the robot could eventually realize turning.

A photo-interrupter (introduced in on-coming chapter) is mounted on top of each motor to monitor its rotation. Motor controller uses H-bridge to control the forward/ reverse rotation of the motor. To achieve certain speed in different waling gaits (e.g. turning and recovering maneuver), Epiphany board sends a modulated square wave (PWM) to motor controller to tune motor speed.

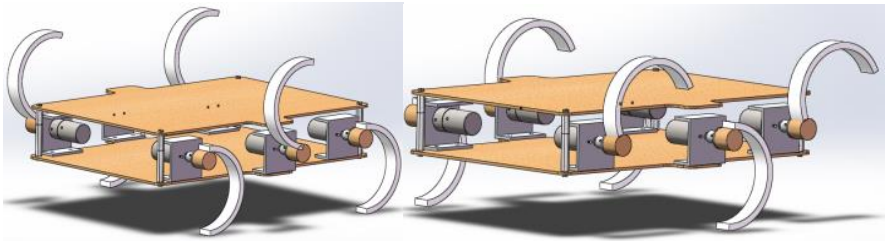

Fig.3 Three-by-three walking gait

The first four motors are controlled by the quad motor controller package on Tim's board. Basically one just defines the motor channel, direction and PWM duty cycle and it is ready to go. For the remaining two motors, I bought a dual motor driver from Pololu.com, and it has similar structure as the quad package on Tim's board. All I did was modifying pre-defined codes to extend the functionality of motor control functions.

The battery for motors that I chose is Thunder Power 3-cell Lipo battery pack with 30C discharge rate and 1800mAh capacity. Each cell of a Lipo battery has a voltage of 3.7V, so a 3s Lipo has a voltage of 11.1V, which will reach 12.6V when fully charged. Since I need my robot to be lightweight, and traditional battery usually accounts for a significant portion of the overall

weight (like NiMH), a Lipo battery seems to be a good choice. I saved 50% of the battery weight by changing it from NiMH rechargeable batteries to Lipo battery. Another reason why I chose Lipo is because my robot

drives six 12V DC motors at the same time, which means that the total current it might draw is very high. A Lipo battery with 30C discharge rate would maximize the performance of motors.

Sensors

Two IR sensors are mounted in front of Hex Guard to serve as the object-avoidance sensor. Since Hex Guard can climb over small obstacles, it reduces the number of sensors necessary to perform object avoidance. Two Sharp® IR rangefinders, which are connected to ADC channels on board, are used to prevent collision with large objects.

When selecting sensors for obstacle avoidance, it is common to come up with bumpers, sonars and IRs. The reason why I chose IR sensor instead of sonar is because IR sensor shoots parallel beams while sonar generates ultrasonic waves in the shape of a cone. Since a walking robot like Hex Guard will often incline forward due to terrain characteristics, sonar might recognize the ground as obstacle, which will cause the robot to stop for no reason. Thus IR sensors are preferable since they can focus only on objects in front of the robot.

Infrared photo-interrupters are mounted on top of motor shaft to serve as simple encoders. Each photo-interrupter has a pair of opposing emitter and detector in a case, providing non-contact sensing. Every time motor shaft rotates one revolution, a piece of plastic mounted on the shaft would ‘interrupt’ the infrared light between the emitter and receiver, producing a low analog reading. In this way the MCU could tell whether either group of legs has completed rotation or not. Below is a picture of photo-interrupter and its schematic diagram.

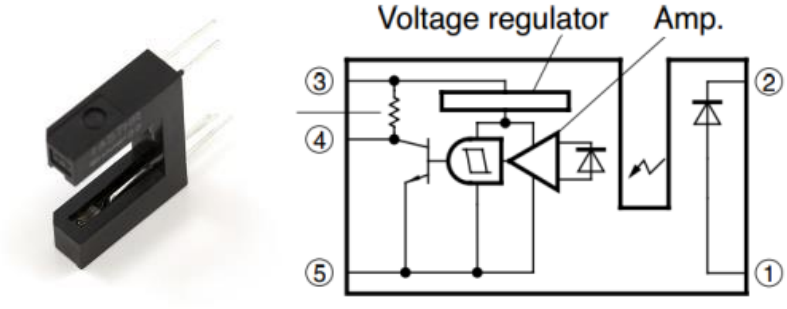

Fig.4 Photo-interrupter

A Velleman K7102 metal detector is used to detect metals. It has a detection range of up to 8cm and it requires a 9V power supply. The metal detector is working on the theory of beat frequency oscillation. The detecting area is the metal stick that is winded with copper wires. I bought a kit of separate components and soldered them into a complete sensor. Then I modified its circuit to realize controlling and communication with the metal detector. Once it detects metal object beneath the robot, Hex Guard stops walking and blinks its LEDs to warn people of potential hazards.

Behaviors

Hex Guard has two primary walking gaits: Three-by-three with close-loop control and six-by-six with open-loop control. The former is intended on smooth surfaces with metal detecting function activated, while the latter is a simple demonstration of how well Hex Guard could maneuver on different terrains. Once the robot is powered, it automatically stands up by rotating all six legs that will simultaneously stop at vertical position. Then it defaults to stand-by mode. At this time, further command from human must be given in order to trigger either walking gaits.

If the left IR is blocked for a certain amount of time, then the three-by-three walking gaits is activated. At this point the robot starts a sequence walking gait as described in "Actuation". A blue LED as well as a red LED on top of the robot indicates different behaviors: While it is marching forward, the blue LED is turned on while the red one is off. While it is reversing, the robot will turn on red LED while turn off the blue one. The robot randomly walks in the area to search for a metal without running into any large obstacles. Once the robot detects a metal objects lying beneath, it will stop moving and blink both LEDs until the metal is removed.

On the other hand, if the right IR sensor is blocked, the robot will activate six-by-six walking gaits by setting all motors run constantly. In this mode the robot will try to climb any obstacles in front of it. When it fails to climb something and gets stuck, it will automatically perform a recover maneuver and randomly turns into other directions to continue the gait.

Experimental Layout and Results

Experiment with Different Close-loop Methods

The most challenging part might be the close loop control of motors. Servos and stepper motors are controlled by sending incremental values to the control channel, which guarantees precise control without any feedback. DC motors, however, requires shaft encoders to ensure precise actuation. Since my robot runs all six legs with DC motor, it would be ideal to have an encoder mounted on the motor. However, 12V DC high-torque geared motors with shaft encoders are extremely expensive, which is definitely not suitable for hobby purpose. The DC motors I use on Hex Guard come without any encoder or place to mount encoder. So I have to figure out some method to realize the close-loop control.

Cds cell

I intended to use photo-resistor to sense between dark and bright colors on the hub (Fig.5), however it turned out that photo-resistor is not sensitive to different colors. After some simple experiments, I gave up this method.

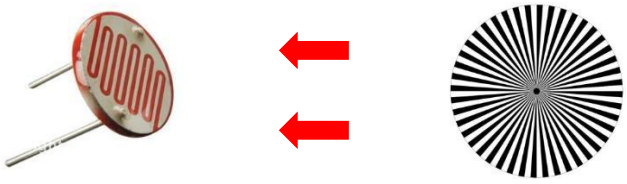

Fig.5 Photo-resistor

Bare wire

Later I tried to use a bare wire to touch the hub to sense a full revolution. First the hub is connected to ground. Then I attached a small piece of tape on the hub. The bare wire was originally pulled low and read into an I/O pin. Every time the wire touches the tape, the signal is pulled high so the MC could tell. The biggest problem with this method is that I could hardly

find a reliable method to mount the wire. Also the contact sensing makes both the wire and tape easily wear out. After some trial and failures, I started to look for some better solutions.

Light gate

Compared to contact sensing, light gate is much more reliable since it requires no contact. Frequently used in printers to sense moving objects, photo-interrupters are ideal for sensing the rotations of motors. After experimenting with it, I found that photo-interrupters will hardly be affected by ambient light due to its embedded design and the infrared characteristics. In addition, the system's response time is still very short even with four ADC channels processing six analog inputs virtually at the same time.

Experiment with Different Structures

Another major issue is balance. DC motors have no self-locking mechanism, although high gear ratio would give some resistance torque when motor is not powered, the whole platform is still very sensitive to perturbations caused by robot locomotion. To stabilize the three legs in static state while other legs are rotating, I did the following design changes: First I lowered the mass center of the robot to reduce perturbations. Second the leg was extended to give more contact area with the ground if it deviates from the equilibrium point. Finally the robot was able hold its position while walking.

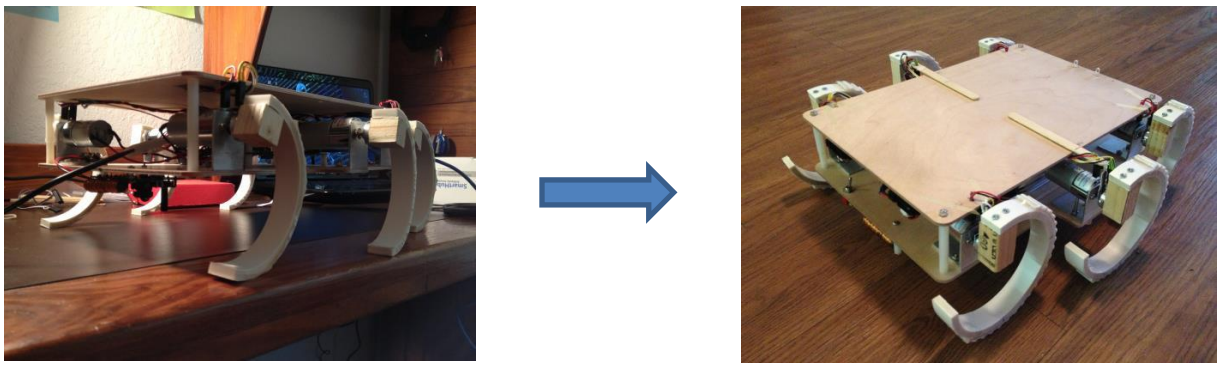

Fig.6 Mass Center Lowered & Leg Extended

Conclusion

IMDL is my first time to build a robot out of nothing. It is truly a great opportunity to gain hands-on experience. At first I was intended to build a robot with many fancy features. However it turned out to be very difficult to realize even basic walking gaits. As a mechanical engineering student, I learned lots of electrical and computer engineering stuff through IMDL, which supplemented huge amount of knowledge that is essential to completely design a robot or a machine. It allows me to access various kinds of sensors and electrical components. In the future, I might want to make the following improvements:

- Design encoders with more ticks, which would add self-locking feature through software
- Replace current DC motors with higher powered ones and more gear ratio
- Mount IP camera onto the robot and play with face recognition and autonomous navigation

References

- Boston Dynamics RHex terrain robot: http://www.youtube.com/watch?v=ISznqY3kESI
- University of Pennsylvania Kod Lab X-RHex technical report: http://kodlab.seas.upenn.edu/uploads/Main/xrhextechreport.pdf
- Select the Right Battery for Your Robot DC: http://vsblogs.wordpress.com/2012/02/09/select-the-right-battery-for-your-robot-dc-motors-part-1-of-2/
- Thunder Power Lithium Polymer Battery: http://www.ebay.com/itm/251084890508?ssPageName=STRK:MEWNX:IT&_trksid=p3984.m1497.l2649
- TB6612FNG Dual Motor Driver: http://www.pololu.com/catalog/product/713
- The code for this project can be found at: https://sites.google.com/site/imdlhex/

Appendices

Project Code

```
/*****************************************************************
Main Function Set Up

 Description: This program set up all initializations and port definitions
*****************************************************************/

\#include <avr/io.h>
#include <util/delay.h>
#include <math.h>

#include "clock.h"
#include "ATtinyServo.h"
#include "uart.h"
#include "adc.h"
#include "motor.h"
#include "RTC.h"

int main(void)
{
	clockInit();
	RTC_DelayInit();// initializing Real Time Clock
	adcInit(&ADCA); // initializing ADC on PORTA
	adcInit(&ADCB); // initializing ADC on PORTB
	ATtinyServoInit();
	usartInit(&USARTC0,115200);
	motorInit();
	sei();

	TCE0.PER = 1024;
	TCE0.CTRLA = TC_CLKSEL_DIV1_gc;
	TCE0.CTRLB = TC0_CCAEN_bm | TC_WGMODE_SS_gc;

	PORTC_DIRSET=0x33; //Accelerometer & LED indicator
	PORTC_OUT=0x01;
	PORTD_OUT=0x1A; // PORTD: 00011010; Enable motor driver standby & drive motor
5,6
	PORTE_DIRSET=0x01; // Set Buzzer pin as output

   stdout = &USB_str; //stdout --> USB : printf writes to the USB port.
```

```
//Setup ADC Mux Channels
        adcChannelMux(&ADCA,2,4); //channel 2 of ADCA will convert from the source on
pin4
        adcChannelMux(&ADCA,3,5);
        adcChannelMux(&ADCB,0,0);
        adcChannelMux(&ADCB,1,1);
        int leg_threshold = 500; // Photo-interrupter reads low when light is blocked
        int metal_threshold = 3900; // Metal Detector reads a value greater than 3900 when metal
is in vicinity
        int IR_threshold = 1800; // IR reads a value greater than 2000 when something is in front
of it


        // walking gait flag
        int three = 0; // three_by_three walking gait
        int three_stuck = 0; // three_by_three stuck flag
        int six = 0; // six_by_six walking gait
        int six_stuck = 0; // six_by_six stuck flag
        int recover_status = 0; // recover stance complete or not
```

```
/*****************************************************************
Stand_up subroutine

 Description: This program performs the stands up
*****************************************************************/

void stand_up() // define stand_up
        {
                setMotorEffort(1,520, MOTOR_DIR_FORWARD);
                setMotorEffort(2,500, MOTOR_DIR_FORWARD);
                setMotorEffort(3,500, MOTOR_DIR_FORWARD);
                setMotorEffort(4,520, MOTOR_DIR_FORWARD);
                setMotorEffort(5,500, MOTOR_DIR_FORWARD);
                setMotorEffort(6,500, MOTOR_DIR_FORWARD);

                int mark=0;// to get rid of unwanted initial value of photo-interrupter
                int16_t m1_state = 1; // flag 1 when motor 1 is running
                int16_t m2_state = 1; // flag 1 when motor 2 is running
                int16_t m3_state = 1; // flag 1 when motor 3 is running
                int16_t m4_state = 1; // flag 1 when motor 4 is running
                int16_t m5_state = 1; // flag 1 when motor 5 is running
                int16_t m6_state = 1; // flag 1 when motor 6 is running
                while (1)
                {
                        mark++;
                        adcChannelMux(&ADCA,0,0);_delay_ms(1);int16_t m5 =
analogRead(&ADCA,0);
                        adcChannelMux(&ADCA,0,1);_delay_ms(1);int16_t m2 =
analogRead(&ADCA,0);
                        adcChannelMux(&ADCA,1,2);_delay_ms(1);int16_t m3 =
analogRead(&ADCA,1);
                        adcChannelMux(&ADCA,1,3);_delay_ms(1);int16_t m6 =
analogRead(&ADCA,1);
                        int16_t m1 = analogRead(&ADCA,2);
                        int16_t m4 = analogRead(&ADCA,3);

                        if (m1 < leg_threshold && m1_state == 1 && mark>1)
{setMotorEffort(1,800, MOTOR_DIR_BACKWARD);_delay_ms(10);setMotorEffort(1,0,
MOTOR_DIR_BACKWARD);m1_state = 0;}// When leg1 reaches its position, brake motor1
                        if (m2 < leg_threshold && m2_state == 1 && mark>1)
{setMotorEffort(2,800, MOTOR_DIR_BACKWARD);_delay_ms(20);setMotorEffort(2,0,
MOTOR_DIR_BACKWARD);m2_state = 0;}// When leg2 reaches its position, brake motor2
                        if (m3 < leg_threshold && m3_state == 1 && mark>1)
{setMotorEffort(3,800, MOTOR_DIR_BACKWARD);_delay_ms(20);setMotorEffort(3,0,
MOTOR_DIR_BACKWARD);m3_state = 0;}// When leg3 reaches its position, brake motor3
                        if (m4 < leg_threshold && m4_state == 1 && mark>1)
```

```
{setMotorEffort(4,800, MOTOR_DIR_BACKWARD);_delay_ms(10);setMotorEffort(4,0,
MOTOR_DIR_BACKWARD);m4_state = 0;}
                        if (m5 < leg_threshold && m5_state == 1 && mark>1)
{setMotorEffort(5,800, MOTOR_DIR_BACKWARD);_delay_ms(20);setMotorEffort(5,0,
MOTOR_DIR_BACKWARD);m5_state = 0;}
                        if (m6 < leg_threshold && m6_state == 1 && mark>1)
{setMotorEffort(6,800, MOTOR_DIR_BACKWARD);_delay_ms(20);setMotorEffort(6,0,
MOTOR_DIR_BACKWARD);m6_state = 0;}
                        if (m1_state == 0 && m2_state == 0 && m3_state == 0 && m4_state ==
0 && m5_state == 0 && m6_state == 0 && mark>1) {break;} // When all legs reach their
position, jump out
                        }
        }
```

```
/*****************************************************************
Emergency stop subroutine

 Description: This program performs emergency stop when metal is in vicinity
*****************************************************************/
void emergency_stop()
{
	PORTC_OUTCLR = 0x20; // Turn off red LED
	PORTC_OUTSET = 0x10; // Turn on blue LED
	int i = 0;

	while (1)
	{

		int16_t metal = analogRead(&ADCB,1);
		if (metal < metal_threshold){i++;}
			if ( i == 10){break;}
		//printf("metal=%d\r",metal);

		setMotorEffort(1,0, MOTOR_DIR_NEUTRAL);
		setMotorEffort(2,0, MOTOR_DIR_NEUTRAL);
		setMotorEffort(3,0, MOTOR_DIR_NEUTRAL);
		setMotorEffort(4,0, MOTOR_DIR_NEUTRAL);
		setMotorEffort(5,0, MOTOR_DIR_NEUTRAL);
		setMotorEffort(6,0, MOTOR_DIR_NEUTRAL);

		PORTC_OUTTGL = 0x30; // Blink both LEDs
		_delay_ms(100);
	}
}
```

```
/*****************************************************************
Reverse_three_by_three subroutine

 Description: This program performs reverse action of robot
*****************************************************************/

void reverse_three_by_three(){
PORTC_OUTCLR = 0x10; // Turn off blue LED
PORTC_OUTSET = 0x20; // Turn on red LED

setMotorEffort(1,700, MOTOR_DIR_BACKWARD);int16_t m1_state = 1;
setMotorEffort(2,700, MOTOR_DIR_BACKWARD);int16_t m2_state = 1;
setMotorEffort(5,650, MOTOR_DIR_BACKWARD);int16_t m5_state = 1;

setMotorEffort(3,200, MOTOR_DIR_FORWARD);
setMotorEffort(4,200, MOTOR_DIR_FORWARD);

int16_t m3_state = 0;
int16_t m4_state = 0;
int16_t m6_state = 0;
_delay_ms(200);

int i = 0; // count 3 cycles then jump out of "reverse" subroutine

while (1)
{
	/******Stuck Prevention******/
	if (delayOver == 1){recover_stance();}

	if (i == 3){recover_stance();break;} // perform "reverse" for three times

	adcChannelMux(&ADCA,0,0);_delay_ms(1);int16_t m5 = analogRead(&ADCA,0);
	adcChannelMux(&ADCA,0,1);_delay_ms(1);int16_t m2 = analogRead(&ADCA,0);
	int16_t m1 = analogRead(&ADCA,2);
	if (m1 < leg_threshold && m1_state == 1) {setMotorEffort(1,1000,
MOTOR_DIR_FORWARD);_delay_ms(20);setMotorEffort(1,200,
MOTOR_DIR_FORWARD);m1_state = 2;}
	if (m2 < leg_threshold && m2_state == 1) {setMotorEffort(2,1000,
MOTOR_DIR_FORWARD);_delay_ms(20);setMotorEffort(2,200,
MOTOR_DIR_FORWARD);m2_state = 2;}
	if (m5 < leg_threshold && m5_state == 1) {setMotorEffort(5,1000,
MOTOR_DIR_FORWARD);_delay_ms(20);setMotorEffort(5,0,
MOTOR_DIR_FORWARD);m5_state = 2;}
	if (m1_state == 2 && m2_state == 2 && m5_state == 2 && m3_state == 0 && m4_state
== 0 && m6_state == 0){
	_delay_ms(1000);
```

```
        setMotorEffort(3,700, MOTOR_DIR_BACKWARD);m3_state = 1;setMotorEffort(4,750,
MOTOR_DIR_BACKWARD);m4_state = 1;setMotorEffort(6,650,
MOTOR_DIR_BACKWARD);m6_state = 1;m1_state = 0; m2_state = 0; m5_state =0;
        _delay_ms(150);
        }

        adcChannelMux(&ADCA,1,2);_delay_ms(1);int16_t m3 = analogRead(&ADCA,1);
        adcChannelMux(&ADCA,1,3);_delay_ms(1);int16_t m6 = analogRead(&ADCA,1);
        int16_t m4 = analogRead(&ADCA,3);

        if (m4 < leg_threshold){RTC_Delay_ms(5000);} //Using real time clock to count the
time that leg gets stuck
        if (m3 < leg_threshold && m3_state == 1) {setMotorEffort(3,1000,
MOTOR_DIR_FORWARD);_delay_ms(20);setMotorEffort(3,200,
MOTOR_DIR_FORWARD);m3_state = 2;}
        if (m4 < leg_threshold && m4_state == 1) {setMotorEffort(4,1000,
MOTOR_DIR_FORWARD);_delay_ms(20);setMotorEffort(4,200,
MOTOR_DIR_FORWARD);m4_state = 2;}
        if (m6 < leg_threshold && m6_state == 1) {setMotorEffort(6,1000,
MOTOR_DIR_FORWARD);_delay_ms(20);setMotorEffort(6,0,
MOTOR_DIR_FORWARD);m6_state = 2;}
        if (m3_state == 2 && m4_state == 2 && m6_state == 2 && m1_state == 0 && m2_state
== 0 && m5_state == 0 ){
        _delay_ms(1000);
        setMotorEffort(1,700, MOTOR_DIR_BACKWARD);m1_state = 1;setMotorEffort(2,700,
MOTOR_DIR_BACKWARD);m2_state = 1;setMotorEffort(5,650,
MOTOR_DIR_BACKWARD);m5_state = 1;m3_state = 0; m4_state = 0; m6_state =0;
        _delay_ms(150);
        i++;
        }
}
}
```

```
/*****************************************************************
Turn_right subroutine

 Description: This program performs right turn action of robot
*****************************************************************/

void turn_right(){
	PORTC_OUTCLR = 0x20; // Turn off red LED
	PORTC_OUTSET = 0x10; // Turn on blue LED

	setMotorEffort(1,750, MOTOR_DIR_FORWARD);int16_t m1_state = 1;
	setMotorEffort(2,750, MOTOR_DIR_FORWARD);int16_t m2_state = 1;
	setMotorEffort(5,750, MOTOR_DIR_BACKWARD);int16_t m5_state = 1;
	_delay_ms(200);
	int i = 0;		// counter
	setMotorEffort(3,300, MOTOR_DIR_FORWARD);
	setMotorEffort(4,300, MOTOR_DIR_FORWARD);
	RTC_Delay_ms(10000); //Using real time clock to limit turning time

	while (1)
	{
	/******Stuck Prevention******/
	if (delayOver == 1){recover_stance();break;}

	adcChannelMux(&ADCA,0,0);_delay_ms(1);int16_t m5 = analogRead(&ADCA,0);
	adcChannelMux(&ADCA,0,1);_delay_ms(1);int16_t m2 = analogRead(&ADCA,0);
	int16_t m1 = analogRead(&ADCA,2);

	if (m1 < leg_threshold && m1_state == 1) {setMotorEffort(1,800,
MOTOR_DIR_BACKWARD);_delay_ms(20);setMotorEffort(1,0,
MOTOR_DIR_BACKWARD);m1_state = 2;}
	if (m2 < leg_threshold && m2_state == 1) {setMotorEffort(2,800,
MOTOR_DIR_BACKWARD);_delay_ms(20);setMotorEffort(2,0,
MOTOR_DIR_BACKWARD);m2_state = 2;}
	if (m5 < leg_threshold && m5_state == 1){setMotorEffort(5,800,
MOTOR_DIR_FORWARD);_delay_ms(20);setMotorEffort(5,0,
MOTOR_DIR_FORWARD);m5_state = 2;}
	if (m1_state == 2 && m2_state == 2 && m5_state == 2){
		i++;
	if (i==5){recover_stance();break;}
		_delay_ms(1000);
		setMotorEffort(1,700, MOTOR_DIR_FORWARD);m1_state =
1;setMotorEffort(2,700, MOTOR_DIR_FORWARD);m2_state = 1;setMotorEffort(5,700,
MOTOR_DIR_BACKWARD);m5_state = 1;
		_delay_ms(150);
	}}}
```

```
/*****************************************************************
Forward_three_by_three subroutine

 Description: This program performs forward three_by_three walking gait
*****************************************************************/
void forward_three_by_three(){
PORTC_OUTCLR = 0x20; // Turn off red LED
PORTC_OUTSET = 0x10; // Turn on blue LED

setMotorEffort(1,700, MOTOR_DIR_FORWARD);int16_t m1_state = 1;
setMotorEffort(2,700, MOTOR_DIR_FORWARD);int16_t m2_state = 1;
setMotorEffort(5,650, MOTOR_DIR_FORWARD);int16_t m5_state = 1;

setMotorEffort(3,100, MOTOR_DIR_BACKWARD);
setMotorEffort(4,100, MOTOR_DIR_BACKWARD);

int16_t m3_state = 0;
int16_t m4_state = 0;
int16_t m6_state = 0;
_delay_ms(200);

int metal_counter = 0;
int IR_counter = 0;

while (1)
{
        /******Metal Detector Interruption******/
        int16_t metal = analogRead(&ADCB,1);
        if (metal > metal_threshold){metal_counter++;}
        if (metal_counter == 10){three = 1; break;} // If metal detector senses metal for ten times,
then break the forward loop */

        /******IR Interruption******/
        adcChannelMux(&ADCB,0,2);_delay_ms(1);int16_t IR_right = analogRead(&ADCB,0);
        adcChannelMux(&ADCB,0,3);_delay_ms(1);int16_t IR_left = analogRead(&ADCB,0);
        printf("IR_right=%d,IR_left=%d\r",IR_right,IR_left);
        if (IR_right > IR_threshold || IR_left > IR_threshold) {IR_counter++;}
        if(IR_counter == 10) {three = 1;break;}

        /******Stuck Prevention******/
        if (delayOver == 1)
        {three_stuck=1;break;}

        /******Sequence Walking******/
        adcChannelMux(&ADCA,0,0);_delay_ms(1);int16_t m5 = analogRead(&ADCA,0);
        adcChannelMux(&ADCA,0,1);_delay_ms(1);int16_t m2 = analogRead(&ADCA,0);
```

```
        int16_t m1 = analogRead(&ADCA,2);
        if (m1 < leg_threshold && m1_state == 1) {setMotorEffort(1,1000,
MOTOR_DIR_BACKWARD);_delay_ms(20);setMotorEffort(1,100,
MOTOR_DIR_BACKWARD);m1_state = 2;}
        if (m2 < leg_threshold && m2_state == 1) {setMotorEffort(2,1000,
MOTOR_DIR_BACKWARD);_delay_ms(20);setMotorEffort(2,100,
MOTOR_DIR_BACKWARD);m2_state = 2;}
        if (m5 < leg_threshold && m5_state == 1) {setMotorEffort(5,1000,
MOTOR_DIR_BACKWARD);_delay_ms(20);setMotorEffort(5,0,
MOTOR_DIR_BACKWARD);m5_state = 2;}
        if (m1_state == 2 && m2_state == 2 && m5_state == 2 && m3_state == 0 && m4_state
== 0 && m6_state == 0){
        _delay_ms(1000);
        setMotorEffort(3,750, MOTOR_DIR_FORWARD);m3_state = 1;setMotorEffort(4,750,
MOTOR_DIR_FORWARD);m4_state = 1;setMotorEffort(6,650,
MOTOR_DIR_FORWARD);m6_state = 1;m1_state = 0; m2_state = 0; m5_state =0;
        _delay_ms(150);
        }

        adcChannelMux(&ADCA,1,2);_delay_ms(1);int16_t m3 = analogRead(&ADCA,1);
        adcChannelMux(&ADCA,1,3);_delay_ms(1);int16_t m6 = analogRead(&ADCA,1);
        int16_t m4 = analogRead(&ADCA,3);
        if (m3 < leg_threshold){RTC_Delay_ms(5000);} //Using real time clock to count the
time that leg gets stuck
        if (m3 < leg_threshold && m3_state == 1) {setMotorEffort(3,1000,
MOTOR_DIR_BACKWARD);_delay_ms(20);setMotorEffort(3,100,
MOTOR_DIR_BACKWARD);m3_state = 2;}
        if (m4 < leg_threshold && m4_state == 1) {setMotorEffort(4,1000,
MOTOR_DIR_BACKWARD);_delay_ms(20);setMotorEffort(4,100,
MOTOR_DIR_BACKWARD);m4_state = 2;}
        if (m6 < leg_threshold && m6_state == 1) {setMotorEffort(6,1000,
MOTOR_DIR_BACKWARD);_delay_ms(20);setMotorEffort(6,0,
MOTOR_DIR_BACKWARD);m6_state = 2;}
        if (m3_state == 2 && m4_state == 2 && m6_state == 2 && m1_state == 0 && m2_state
== 0 && m5_state == 0 ){
        _delay_ms(1000);
        setMotorEffort(1,700, MOTOR_DIR_FORWARD);m1_state = 1;setMotorEffort(2,700,
MOTOR_DIR_FORWARD);m2_state = 1;setMotorEffort(5,650,
MOTOR_DIR_FORWARD);m5_state = 1;m3_state = 0; m4_state = 0; m6_state =0;
        _delay_ms(150);
        }
}
}
```

```
/*****************************************************************
Forward_six_by_six subroutine

 Description: This program performs forward six_by_six walking gait
*****************************************************************/
void butterfly_forward(){
                setMotorEffort(1,600, MOTOR_DIR_FORWARD);int16_t m1_state = 1;
                setMotorEffort(2,600, MOTOR_DIR_FORWARD);int16_t m2_state = 1;
                setMotorEffort(3,600, MOTOR_DIR_FORWARD);int16_t m3_state = 1;
                setMotorEffort(4,600, MOTOR_DIR_FORWARD);int16_t m4_state = 1;
                setMotorEffort(5,500, MOTOR_DIR_FORWARD);int16_t m5_state = 1;
                setMotorEffort(6,500, MOTOR_DIR_FORWARD);int16_t m6_state = 1;
                _delay_ms(200);

                int counter = 0;
                int IR_counter = 0;

                while (1)
                {
                        if (delayOver == 1){six_stuck=1;break;} // When leg is stuck for more
than 2 seconds, break while loop and wait for recover stance

                        adcChannelMux(&ADCA,1,2);_delay_ms(1);int16_t m3 =
analogRead(&ADCA,1);
                        if (m3 < leg_threshold) {RTC_Delay_ms(2000);counter++;
_delay_ms(200);} // Every time photo-interrupter reads a value, re-set counting down and add
counter
                        if (counter == 5){six = 1;break;}

                        /******IR Interruption******/
                        adcChannelMux(&ADCB,0,2);_delay_ms(1);int16_t IR_right =
analogRead(&ADCB,0);
                        adcChannelMux(&ADCB,0,3);_delay_ms(1);int16_t IR_left =
analogRead(&ADCB,0);
                        printf("IR_right=%d,IR_left=%d\r",IR_right,IR_left);
                        if (IR_right > IR_threshold || IR_left > IR_threshold) {IR_counter++;}
                        if(IR_counter == 10) {six = 1;break;}
                }
        }
```

```
/****************Metal Detector Interruption**********************/
        int16_t metal = analogRead(&ADCB,1);
        if (metal > metal_threshold){emergency_stop();_delay_ms(500);recover_stance();}
        //printf("metal=%d\r",metal);

/****************IR Interruption************************************/
        adcChannelMux(&ADCB,0,2);_delay_ms(1);int16_t IR_right = analogRead(&ADCB,0);
         adcChannelMux(&ADCB,0,3);_delay_ms(1);int16_t IR_left = analogRead(&ADCB,0);
         //printf("IR_right=%d,IR_left=%d\r",IR_right,IR_left);
        if ((IR_right > IR_threshold || IR_left > IR_threshold) && IR_left >
IR_right ){recover_stance();reverse_three_by_three();turn_right();}
        if ((IR_right > IR_threshold || IR_left > IR_threshold) && IR_left <
IR_right ){recover_stance();reverse_three_by_three();turn_left();}
        //if (IR_right > IR_threshold || IR_left > IR_threshold && six
==1){_delay_ms(1000);butterfly_reverse();}

        if (IR_right < IR_threshold && IR_left < IR_threshold && three == 1)
{forward_three_by_three();} // continue three_by_three gait if previously flagged as three
        if (IR_right < IR_threshold && IR_left < IR_threshold && three_stuck
==1){recover_stance();reverse_three_by_three();turn_left();three_stuck = 0; three = 1;} //
recover stance if robot is stuck during forward_three_by_three gait

        if (IR_right < IR_threshold && IR_left < IR_threshold && six
==1){butterfly_forward();}  // continue six_by_six gait if previously flagged as six
        if (IR_right < IR_threshold && IR_left < IR_threshold && six_stuck
==1){recover_stance();reverse_three_by_three();turn_left();six_stuck = 0; six = 1;}  // recover
stance if robot is stuck during six_by_six gait
```