THE INSTITUTE FOR SYSTEMS RESEARCH

ISR TECHNICAL REPORT 2008-15

Networked Mini-Robots

Chung, Calvin
Advisor: Bergbreiter, Sarah

Networked Mini-Robots

By Calvin Chung

In collaboration with George Gateau and Ivan Penskiy

ISR Summer Research Education for Undergraduate 2008

Advisor: Sarah Bergbreiter

Content

1. Abstract

The mini-robots are designed to develop techniques which can be used to cooperate in multiple robots systems and to distribute sensing robot network in a greater scale. Each mini-robot is entirely constructed of off-the-shelf components. The robot, which is approximate a penny size, is small that every components which the robot implements are tiny. These components provide mobility, radio communication and actuating ability to the robots. The software to control each robot is written in C Code, and the application detail is abstracted away by the developer. A high level human computer interface provides the access for the user to input the command to the robots through the wireless protocol. The software is capable of being extended in a larger-scale system and being implemented into similar types of robots for various purposes.

2. Introduction

Currently, there are many ongoing researches on networked mini-robots. Robots between robots sending information to each other allow them to communicate so that they can know the situation of one another. Information transferring between robots allows them to locate another robot’s position and to communicate. The ultimate goal of this research is to develop a communication interface for the robots. At the beginning of the designing phase, design an interface for the user to input the signal from the computer is essential to test for the wireless availability of the device. After demonstrating the wireless functionality, future developers can modify the software and develop a feedback control loop interface for the robot to communicate by implementing the wireless protocol. Since this project is still at early stage, the task of this research is to device a human computer interface so that the users can input the signal in it. The

signal transfer serially to the wireless access point and sends to the wireless end point of the device. The signal received by the end point actuates the devices and drive the motor.

3. Objective

Since the software and the hardware is designed for a large-scale robots system in future works, all the hardware and components utilized by the robots are necessary to be off-the-shelf for simplicity and convenience. The first objective in this project is to equip the hardware together and to construct mini-robots. This hardware is easy to install, and it is compatible with other components which are commonly found in a robot system. The second objective is to design software to control the motor device in order to provide basic functionality and movement for the robot. The software allows users and the future developers easily to interpret and utilize in the future. The third goal is to design an interface which allows multiple robots to communicate, transfers information and signal through a wireless platform. The project is significantly emphasized on software design for motor and wireless controlling.

4. Designing Platform & Consideration

The eZ430-RF2500 uses the *IAR Embedded Workbench Integrated Development Environment* (IDE) to write, download, and debug an application. The debugger is unobtrusive, allowing the user to run an application at full speed with both hardware breakpoints and single stepping available while consuming no extra hardware resources [1]. The IDE supports different programming language such as C, C++ and Assembly for developers to use in coding. A USB debugging interface is necessary for transferring the code from IDE to the USB device. Considering the convenience for the user to implement the software, developer designs a lower

level machine code which the mechanical detail is abstracted away. Therefore, it allows future developer implementing this software without knowing the peripheral hardware underlying in the program. The reasons to use C Code to write because it is rather simple and easy to decipher, comparing the other alternative such as Java and C++.

5. The Hardware of the Mini-Robots

Hardware of a robot grants different functionality of the robot. The mini-robot equipped with the microcontroller control the hardware device of robot. Two DC motors are used to control the speed of the wheels on right side and the wheels on the left side. Motor controller contains actuate devices which drive the motor and receive the signal from the microcontroller. The H-bridge is a hardware component which is mounted on the Motor Board.

5.1 Microcontroller

The Microcontroller in this project is **MSP430F2274** which is developed by the Texas Instrument. A microcontroller acting like a microprocessor in the computer is responsible for signal processing. It is a programmable device which receives the software code and executes the instruction given by the developer. It combines the small size, low power consumption, and computational abilities of an inexpensive microprocessor with the signal-processing proficiency of discrete circuits. The microcontroller includes such built-in amenities as a serial-line, analog-to-digital convertors, timer, and pulse counter [2].

5.2 Motor Controller

A motor controller is a device or group of devices that serves to govern in some predetermined manner the performance of an electric motor. A motor controller might include a manual or automatic means for starting and stopping the motor, selecting forward or reverse rotation and regulating the speed [3]. The motor controller in the project is mounted with an H-bridge which

drives the functionality on the above. Other actuate device are possibly found on the Motor Controller Board.

5.3 H-Bridge

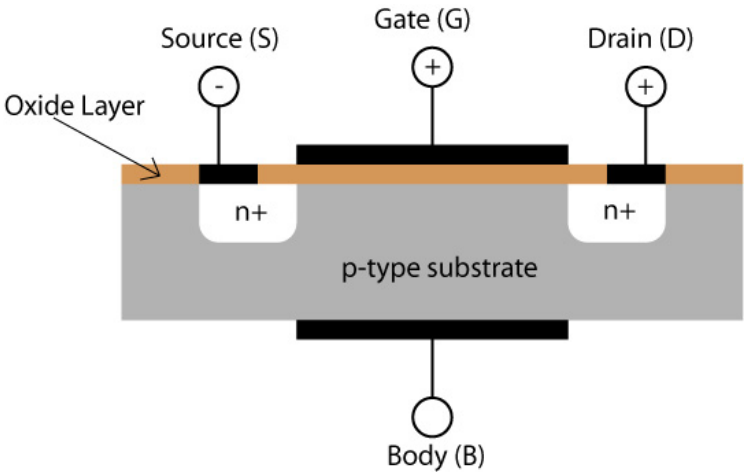


Fig 5.1 The schematics of MOSFET

An H-Bridge is commonly constructed by using reverse polarity device such as **MOSFETs**, shown in **Fig 5.1** or bipolar transistors circuit. Their functionality and topology are similar that both are available for H-Bridge implementation. In a **MOSFET** implementation of an H-bridge, a p-channel device is soldered on the top and n-channel devices on bottom. In a bipolar transistor circuit implementation of an H-bridge, a PNP device is soldered on top and a NPN device on bottom [4]. **MOSFET** devices are implemented in this project.

5.4 DC Motor

DC motors are commonly used in driving movement of a small machine. They also appear in a large variety of shapes and sizes. Since the size of the mini-robots is small, the DC gearhead motors are implemented in this project. DC gearhead motors are normally consist of permanent magnet ironless rotor in order to be as lightweight as possible [5].

6. Speed Control

The principal of the speed control in a digital system is dictated by the signal generated by the microcontroller. This signal is called the Pulse Width Modulation (PWM), which is a common type of signals used in digital real-time control. The timer and clock in the processor synchronize the PWM signals, generate various wave frequency and different length of dutycycle. The dutycycles of the PWM signals governed the voltage output, and thus the speed of the motor.

6.1 Timer

The microcontroller used in this project, **MSP430F2274**, contains two different timers in the processor. The Timer A and Timer B works similar and contains exactly the same set of instruction in their registers. However, each timer only output the signals in some pins. The directories of these pins are defined by the system during the manufacture of the microcontroller. These Timers also contain a timer counter which counts the value ranging from 0 to 2^{16}. Both Timer A and Timer B contain three clocks in the system which is the Auxiliary Clock (ACLK), the Master Clock (MCLK), and the Sub-Main Clock (SMCLK). Each clock can be calibrated by four different mode of frequency. Moreover, they contain a pre-scalar (divider) to scale the period time. The Timer A and Timer B have a register to select the clocks, the value which the timer counter counts to and the output operation of the signal. The principal for selecting the instruction of the register is to select the bits which represent the instruction you wish to choose. The **TACTL** register allows developers to select the clocks, the pre-scalars and the operation mode of the timer counter. The **TACCR0** register selects what value the timer counter count up to, and execute the output operation of the signal at this value. Developer can select the output operation of the signals through the register **TACCTLx** in the timer.

6.2 PWM signals

Developer have to control the PWM signals in order to control the speed of the motor. The PWM signal is synchronized with the timer counter and the clock signals. In this project, SMCLK is selected, calibrated into 1MHz ensuring that the timer counter will increment on every 10^{-6}s. The

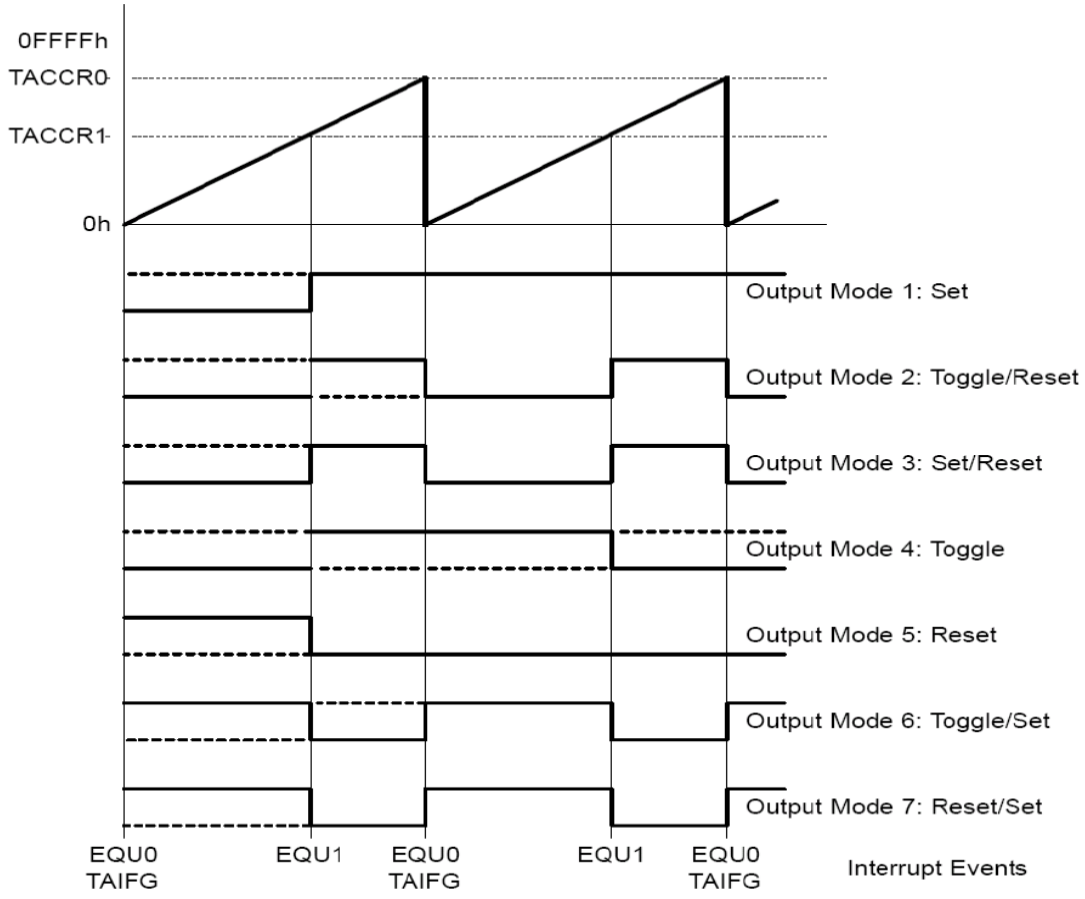


Fig 6.1 Seven operation mode and operation mode of the timer (Up Mode)

TACTL register selects the operation mode of the timer counter in this project to be Up Mode which the timer counter counts to the value setting in **TACCR0.** When the timer counter count to the value of TACCR0, it draws back to 0 and repeat the process as shown in **Fig 6.1**.

TACCTL1 register sets the operation mode of the PWM signals as reset/set, trigger the timer will reset the signal at **TACCR1** and set the signal at **TACCR0**. For examples, if the value of **TACCR1** and **TACCR0** in the project is set to be 5000 and 10000 respectively, the timer will reset the signal at 5000 and set it at 10000.

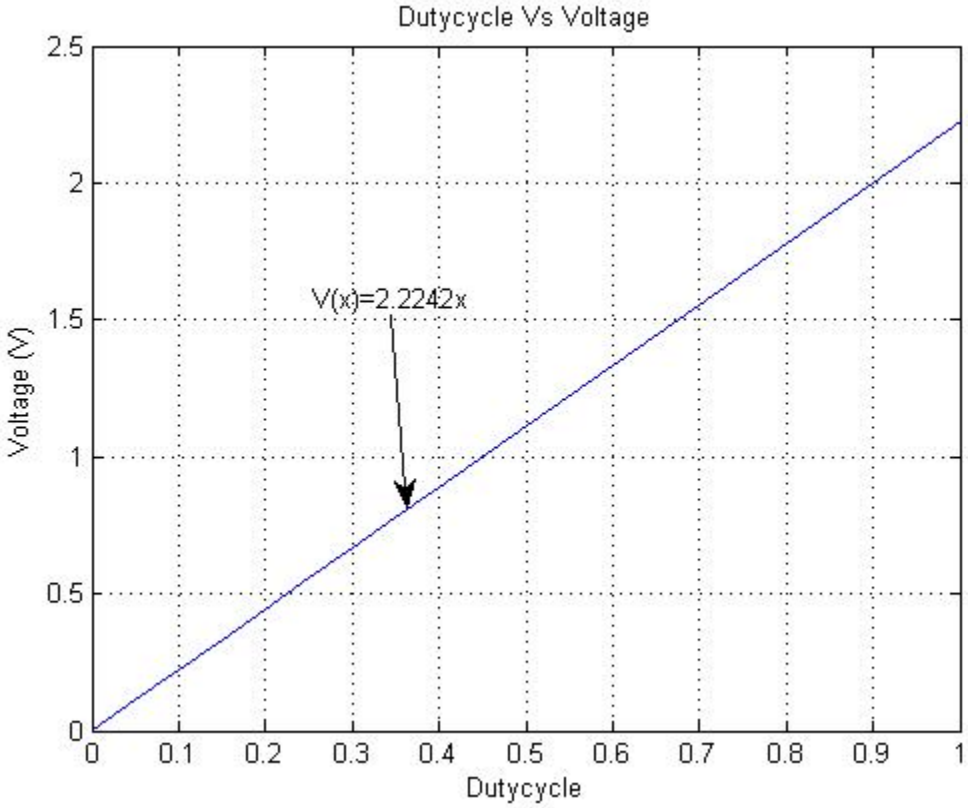


Fig 6.2 The Dutycycle Vs Voltage graph with maximum voltage 2.23V

Therefore, the frequency of the PWM signals is $10^{-6} \times 10000 = 0.01$s. Since the frequency of the signal is high, it is reasonable to regard a discrete signal as a continuous signal, and the voltage is equivalent to the RMS voltage. The dutycycle of the PWM is 0.5. This implies that the speed of

the motor runs at 50% of its maximum speed, since the voltage is proportional to the dutycycle, shown in **Fig 6.2**. The equation

$$\textbf{Dutycycle} = \textbf{TACCR1} \div \textbf{TACCR0}$$

determines the dutycycle of the PWM signals.

7. Direction Control

The direction control of the robots is governed by the intricate cooperation between the H-bridge and the DC motors. The H-Bridge is constructed by using the reverse polarity devices. It receives the signal and determines the polarity based on whether the signal goes high or low relative to another side of the H-bridge. The polarity of the circuit determines the turning orientation of the motor, and thus the direction of the movement of the robot. The orientation of the current plays an important role in the direction control of the robot. Therefore, H-bridge functioning as four switches devices controls the flowing direction of the current, and it is implemented to control the direction of the movement.

7.1 Functionality of the H-bridge

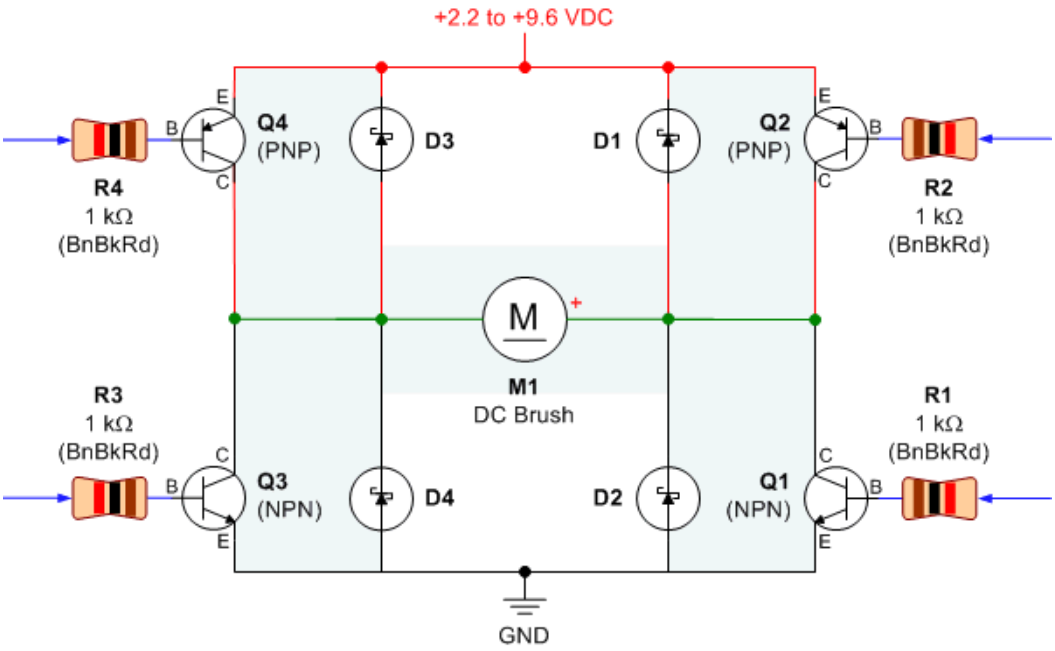


Fig 7.1 The schematic of a Bipolar Transistor H-Bridge

H-Bridge mounted on the motor controller receiving the signal from the microcontroller controls the motion and the movement of the robot. This signal generated by the microcontroller is weak that unable to drive the motor. Therefore, an amplifier circuit is necessary to amplify the signal in order to drive the motor. H-Bridge is implemented in a sense acting as an amplifier.

The H-Bridge shown in **Fig 7.1** is built by four bipolar transistors. Each transistors work like a switch that whether it close or not is dictated by the PWM signal going high or low. For example, according to **Fig 7.1**, when the left side of the H-Bridge goes high and the right side of the H-Bridge goes low, the bipolar transistor of the left side on the H-bridge Q3 will open and Q4 will close, the right side of the bipolar transistors Q1 will close and Q2 open, so the current will flow through Q2 to the motor and exits from Q3 to GND. Therefore, the motor drives the robot to move forward. Reversing the signal with going high on the right and low on the left, the switches react oppositely comparing the situation in forward motion. Therefore, the orientation of the current is reverse, and thus the motor will turn backward.

7.2 Functionality of the DC motor

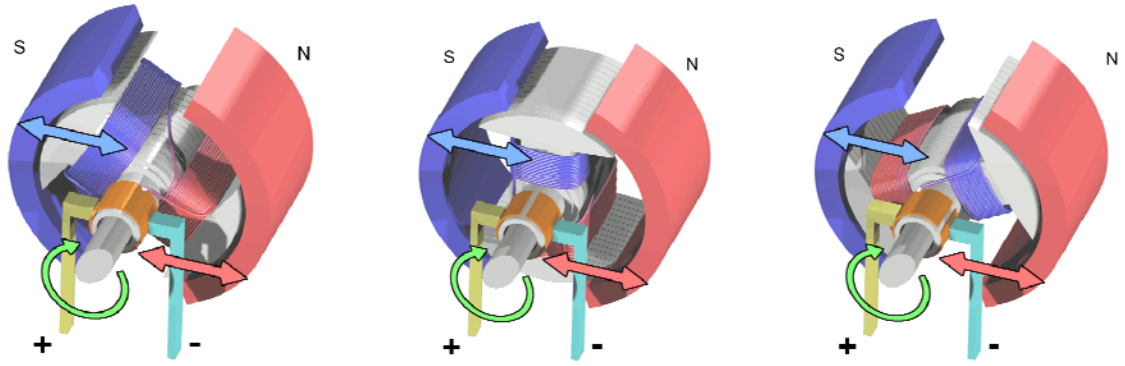


Fig7.2 The schematics of the DC motor with different orientation.

The robot contains two DC motor that each of those controls right side and the left side of the wheels. The DC motor is an electromechanical device which converts electrical energy to mechanical energy, provides dynamical motion and movement to the robot. The principal of DC motor underneath the Lorentz Force Law in electromagnetism states that there is a magnetic force creating on a moving charged object which is under the exposure of magnetic field. Rotary motion requires several loops of wires. **Fig 7.2** shows a loop of wire mounted on an axis of rotation and situated in the flux field set up by the permanent magnets. Because forces are created in a direction in a direction perpendicular to both the current's direction and the magnetic field's direction, current going into the loop along the top generates, according to the left hand rule, a force acting upward. Current coming out along the bottom portion of the loop creates a force acting downward [6]. The two opposite direction forces, acting on each side of the loops, generate a couple moments that cause the rotor rotating clockwise. Continuous rotation can be achieved by reversing the direction of the current and require a device called commutator. A commutation system using brushes is one way to make a DC motor. The commutator segments are attached to the loop of wire a rotate with it, while the brushes remain stationary as the

commutator segments slide past [7]. Therefore, combing the effect of the H-bridge, when the current flowing in from the bottom of the loop the motor will turn anticlockwise, but when the current flowing in from the top of the loop will turn clockwise. Incorporating two device grant the direction of the movement of the robots.

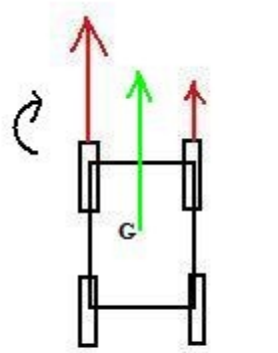


Fig 8.1 The steering Right Mode

8. Turning Control

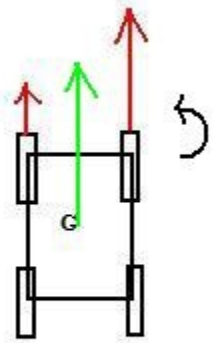


Fig 8.2 The Steering Left Mode

The turning control of the robot is governed by two factors: the velocity of each side of the wheels and the turning orientation of the wheels. There are two DC motors equipped in the robots. Since there are two H-bridge and two timers of the microcontroller in the robots, each H-bridge and each timer control one side of the wheels. Therefore, this structural design allows different velocity and turning orientation applying on each side of the wheels. Four basic modes of turning are programmed in this robot. The first mode as shown in **Fig8.1** is steering right. The key to understand the underlying principal of how it works is to regard the robot body

as a lever structure, the center of gravity in the robot as the pivot of the lever. According to **Fig 8.1**, the velocity vector on the left is greater than that on the right. Applying the principal of levers, the velocity vector on the left side causes the lever having a clockwise rotation tendency. However, the wheels of right side rotate at the same direction with a lower velocity, create an anticlockwise rotation tendency to overcome the clockwise rotation creating by the left side. Therefore, the total velocity on the center of gravity is obtained by the sum of these forward velocity vectors. The angular velocity is obtained by the difference of the velocity multiply the length from the center of gravity to the wheels of the robots. The steering right mode provides the robot not only the rotation but also the forward movement, so the robot steers to the right. The steering left shown in **Fig 8.2**, which shares the same underneath principal, reverse greater velocity on the left to the right side and the lower velocity to the left, cause the robot steer to the left.

Fig 8.3 The Turn Right Mode

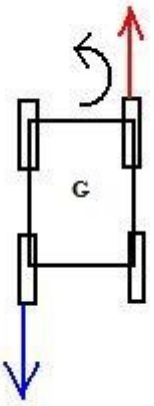
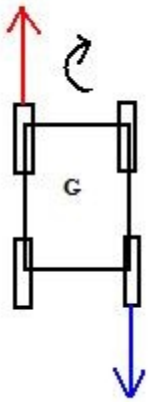


Fig 8.4 The Turn Left mode



The turn right mode, shown in **Fig 8.3**, acting similar to the steer right, but the difference is that it only provides clockwise rotation. The velocity vector on both sides cancels each other because of the opposite direction of movements. However, both vector create clockwise rotation and formed a couple moment. Therefore, the robot turns in place to the right but no forward or backward motion. The turn left mode, shown in **Fig 8.4**, shares the same principal as the turn

right mode. The difference is that the distributions of the velocity vectors are reversed. Therefore, the robot turns in place to the left.

9. Wireless Applications

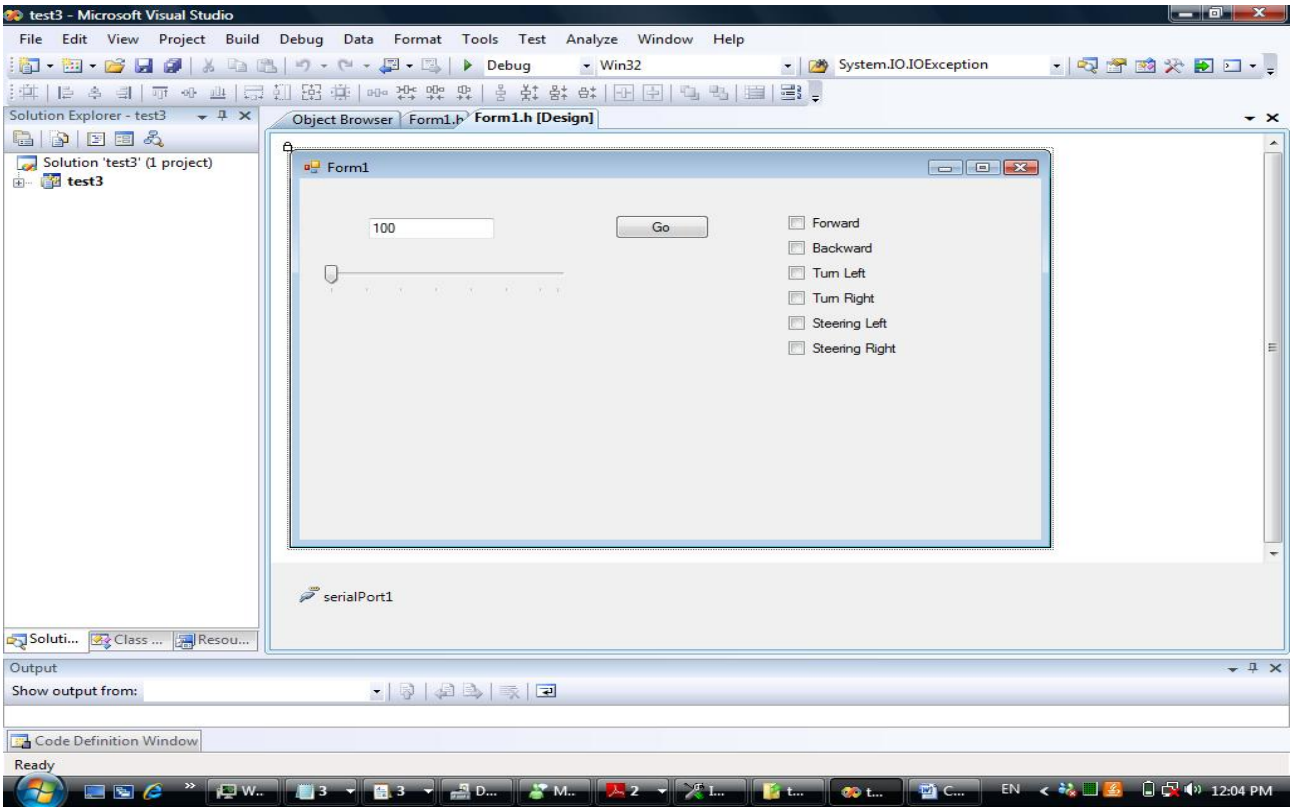


Fig 9.1 User can input the command through this menu

The users input the command through the computer, and the signals transfer from the computer to the robot. In order to accomplish this task, developer needs to build an interface between the computer and the robot. Since the robot is moving around an open space, the wireless applications are necessary to be applied in this project. So that the users can control the robot through the computer, and the robot is able to receive this signal. Three components constitute the wireless protocol: the human computer interface shown in **Fig 9.1**, wireless access point and wireless end point. Users input the commands through the human computer interface, and the signal is transferred to the access point. From the access point, the signal is transmitted though the wireless device and the radio at the end point receive these signals. The microcontroller, **MSP430F2274**, uses a wireless protocol called simpliciTI to perform the transmitting process. The SimplicTI protocol uses **SMPL_Send()** at the access point to send the message, and

SMPL_Receive() to receive the message at the end. Message inputted by the user send serially to the receiving buffer, **UCA0RXBUF**, at the access point. The access point software checks if there is any data in the **UCA0RXBUF**, and trigger the **SMPL_Send()** to send the signal to the end point. The end point receives the message through **SMPL_Receive()**. The simpliciTI protocol allows developer to send a bunch of data in a packet form. The packet defined by the developer stored a set of instructions and message which run the program.

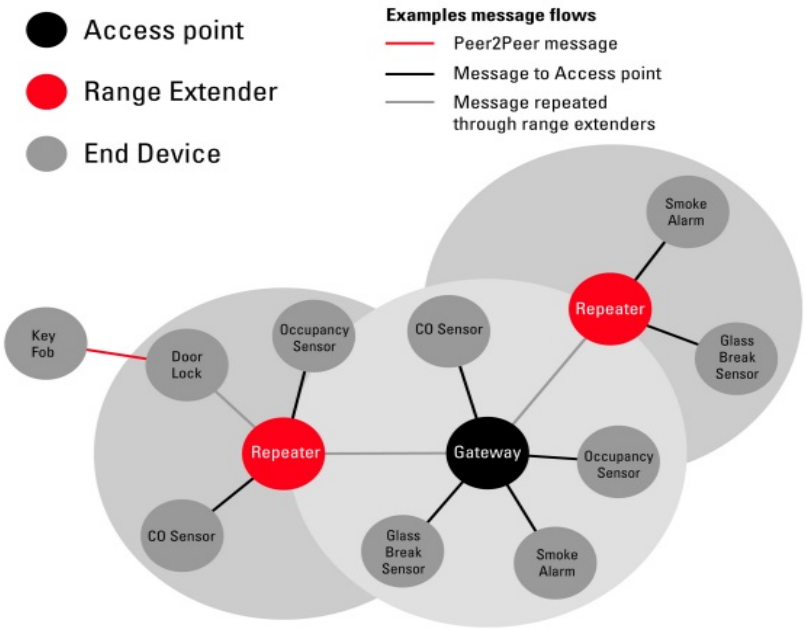


Fig 9.2 The concept of sensor network by using SimpliciTI Protocol

Moreover, the simpliciTI protocol is compatible with the sensor network, can be used as an interface between sensors and actuator to transfer the data and messages as shown in **Fig 9.2**.

10. Future Works

The ultimate goal of this design is to build a networked communication interface for the mini-robots to interact to each other. Instead of using computer sending the signal to demonstrate the wireless function in the microcontroller, future developer utilizes the user interface developed in this project and transforms it into a communication interface for further development. In order to facilitate two boards to be well-communicated, an interface between them is necessarily to be built in future work. The UART or the SPI which is available in the microcontroller are the best to accomplish this task. The design of the networked mini-robot can be applied in some cooperate mission involve severe environmental conditions where human are not able to survive. So the mini-robots can collaborate, simulate the work of human and complete their task. The design of the C Code in this research is well-documented and preserved for future development.

Reference

[1] *Texas Instrument eZ430-RF2500 Development Tool Developmental Tool – User Guide*, http://focus.tij.co.jp/jp/lit/ug/slau227c/slau227c.pdf , June 2008

[2] Inspiration to Implementation second edition – Joseph L. Jones, Bruce A. Seiger and Anita M.Flynn , A K Peter Ltd., Dec,1999, pp49.

[3] Motor controller From Wikipedia, http://en.wikipedia.org/wiki/Motor_controller

[4] Inspiration to Implementation second edition – Joseph L. Jones, Bruce A. Seiger and Anita M.Flynn , A K Peter Ltd., Dec,1999, pp 244

[5] Inspiration to Implementation second edition – Joseph L. Jones, Bruce A. Seiger and Anita M.Flynn , A K Peter Ltd., Dec,1999, pp 194

[6] Inspiration to Implementation second edition – Joseph L. Jones, Bruce A. Seiger and Anita M.Flynn, A K Peter Ltd, Dec, 1999,pp 200

[7] Inspiration to Implementation second edition – Joseph L. Jones, Bruce A.Sieger and Anita M.Flynn, A K Peter Ltd, Dec, 1999 pp202

Code

**********End Point**********

```
**********************************Library ****************************************
#include "msp430x22x4.h"
#include "math.h"
#include "bsp.h"
#include "mrfi.h"
#include "nwk_types.h"
#include "nwk_api.h"
#include "bsp_leds.h"
#include "bsp_buttons.h"
#include "vlo_rand.h"
#include "stdlib.h"
**********************************Function Prototype**********************************
void execute(void);
void set_Motor_A(volatile unsigned int);
void set_Motor_B(volatile unsigned int);
void corner_turn_left(void);
void corner_turn_right(void);
void steering_left(void);
void steering_right(void);
void forward(void);
void backward(void);
void motion(char);
**********************************Global Variable**********************************
#define MESSAGE_LENGTH 6
char signal = '\0';
int direction_A;
int direction_B;
volatile unsigned int speed =500;
linkID_t linkID1;
uint8_t  msg[MESSAGE_LENGTH], len;
**********************************Function Definition**********************************
void main (void)
{
  WDTCTL = WDTPW + WDTHOLD;
  {
    volatile int i;
    for(i = 0; i < 0xFFFF; i++){}
  }

  while (SMPL_NO_JOIN == SMPL_Init((uint8_t (*)(linkID_t))0))
  {

  }
  execute();
}
void execute(void)
{
  uint8_t i, lt;
  volatile unsigned int spd = 0;
  while (SMPL_SUCCESS != SMPL_Link(&linkID1))
  {

  }
```

```
  while (1)
  {
   if(SMPL_Receive(linkID1, msg, &len) == SMPL_SUCCESS)
   {
    if( (msg[0] - 48) != 6)
    {
     speed = 0;
     lt = msg[0] - 48;
     for (i=1; i<lt+1; i++)
     {
      speed = speed*10+(msg[i] - 48);
     }
    }
    else
    {
     signal = msg[1];
     motion(signal);
    }
   }
  }
}

void set_Motor_A(volatile unsigned int number)
{
   if (direction_A == 1)
   {
   P2DIR |= 0x18;
   P2SEL |= 0x18;

      BCSCTL1 = CALBC1_1MHZ;
      DCOCTL = CALDCO_1MHZ;
      TACCR0 = 10000;
      TACCTL1 = OUTMOD_5;
      TACCR1 = 0;
      TACCTL2 = OUTMOD_7;
      TACCR2 =  number;
      TACTL = TASSEL_2 + MC_1;

   }
  if (direction_A == 0)
   {
    P2DIR |= 0x18;
    P2SEL |= 0x18;

      BCSCTL1 = CALBC1_1MHZ;
      DCOCTL = CALDCO_1MHZ;
      TACCR0 = 10000;
      TACCTL1 = OUTMOD_7;
      TACCR1 =  number;
      TACCTL2 = OUTMOD_5;
      TACCR2 = 0;
      TACTL = TASSEL_2 + MC_1;
  }
}
```

```
void set_Motor_B(volatile unsigned int number)
{

 if (direction_B == 0)
 {
 P4DIR |= 0x30;
 P4SEL |= 0x30;

   BCSCTL1 = CALBC1_1MHZ;
   DCOCTL = CALDCO_1MHZ;
   TBCCR0 = 10000;
   TBCCTL1 = OUTMOD_5;
   TBCCR1 = 0;
   TBCCTL2 = OUTMOD_7;
   TBCCR2 = number;
   TBCTL = TBSSEL_2 + MC_1;
 }
  if (direction_B == 1)
 {
 P4DIR |= 0x30;
 P4SEL |= 0x30;

   BCSCTL1 = CALBC1_1MHZ;
   DCOCTL = CALDCO_1MHZ;
   TBCCR0 = 10000;
   TBCCTL1 = OUTMOD_7;
   TBCCR1 = number;
   TBCCTL2 = OUTMOD_5;
   TBCCR2 = 0;
   TBCTL = TBSSEL_2 + MC_1;
 }
}

void forward(void)
{
 direction_A = 1;
 direction_B = 1;
 set_Motor_A(speed);
 set_Motor_B(speed);
}

void backward(void)
{
 direction_A = 0;
 direction_B = 0;
 set_Motor_A(speed);
 set_Motor_B(speed);
}


void corner_turn_left()
{
```

```
 direction_A = 0;
 direction_B = 1;
 set_Motor_A(speed);
 set_Motor_B(speed);
}
void corner_turn_right()
{
 direction_A = 1;
 direction_B = 0;
 set_Motor_A(speed);
 set_Motor_B(speed);
}

void steering_left()
{
 direction_A = 1;
 direction_B = 1;
 set_Motor_A(7000);
 set_Motor_B(speed);
}

void steering_right()
{
 direction_A = 1;
 direction_B = 1;
 set_Motor_A(speed);
 set_Motor_B(7000);
}
void motion(char mode)
{
  switch(mode)
 {

  case 'w':forward();
        break;
  case 's':backward();
        break;
  case 'a':corner_turn_left();
        break;
  case 'd':corner_turn_right();
        break;
  case 'q':steering_left();
        break;
  case 'e':steering_right();
        break;
  default:execute();
  }
}
```

********** **********Access Point**********
**********Library**********

```
#include "bsp.h"
#include "mrfi.h"
#include "bsp_leds.h"
#include "bsp_buttons.h"
#include "nwk_types.h"
#include "nwk_api.h"
#include "nwk_frame.h"
#include "nwk.h"
#include "msp430x22x4.h"
#include "vlo_rand.h"
************************************Function Prototype*********************************
static linkID_t sLID;
static uint8_t sCB(linkID_t);
static uint8_t sJoinSem;
static uint8_t sTransfer;
static uint8_t sConnect;
************************************Global Variable*********************************
unsigned int ni, nn;
************************************Function Definition*********************************
void main (void)
{
  bspIState_t intState;

  WDTCTL = WDTPW + WDTHOLD;
  {
    volatile int i;
    for(i = 0; i < 0xFFFF; i++){}
  }

  BSP_Init();
  sTransfer = 0;
  sConnect = 0;

  BCSCTL1 = CALBC1_8MHZ;
  DCOCTL = CALDCO_8MHZ;

  P3SEL |= 0x30;
  UCA0CTL1 = UCSSEL_2;
  UCA0BR0 = 0x41;
  UCA0BR1 = 0x3;
  UCA0MCTL = UCBRS2;
  UCA0CTL1 &= ~UCSWRST;
  IE2 |= UCA0RXIE;

  __bis_SR_register(GIE);

  SMPL_Init(sCB);
```

```
  while (1)
  {
    if (sJoinSem)
    {
      SMPL_LinkListen(&sLID);
      BSP_ENTER_CRITICAL_SECTION(intState);
      sJoinSem--;
      BSP_EXIT_CRITICAL_SECTION(intState);
      BSP_TURN_ON_LED1();
    }

    if (sTransfer && sConnect)
    {
      BSP_TOGGLE_LED2();
      BSP_ENTER_CRITICAL_SECTION(intState);
      sTransfer--;
      BSP_EXIT_CRITICAL_SECTION(intState);

      SMPL_Send(sLID, mem, sizeof(mem));

      __bis_SR_register(GIE);
     }
  }
}
static uint8_t sCB(linkID_t lid)
{
  if (lid)
  {
    //sPeerFrameSem++;
  }
  else
  {
    sJoinSem++;
    sConnect++;
  }.
  return 0;
}
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
  char rx = UCA0RXBUF;
  if  (UCA0RXBUF != 0x0A)
  {
    mem[ni] = rx;
    ni++;
    if ((mem[0] - 48) == 6)
    {
      nn++;
```

```
    }
    if ((ni == (mem[0] - 48 + 1)) || (nn == 2))
    {
      ni = 0;
      nn = 0;
      sTransfer++;
      __bic_SR_register_on_exit(GIE);
    }
  }
}
```