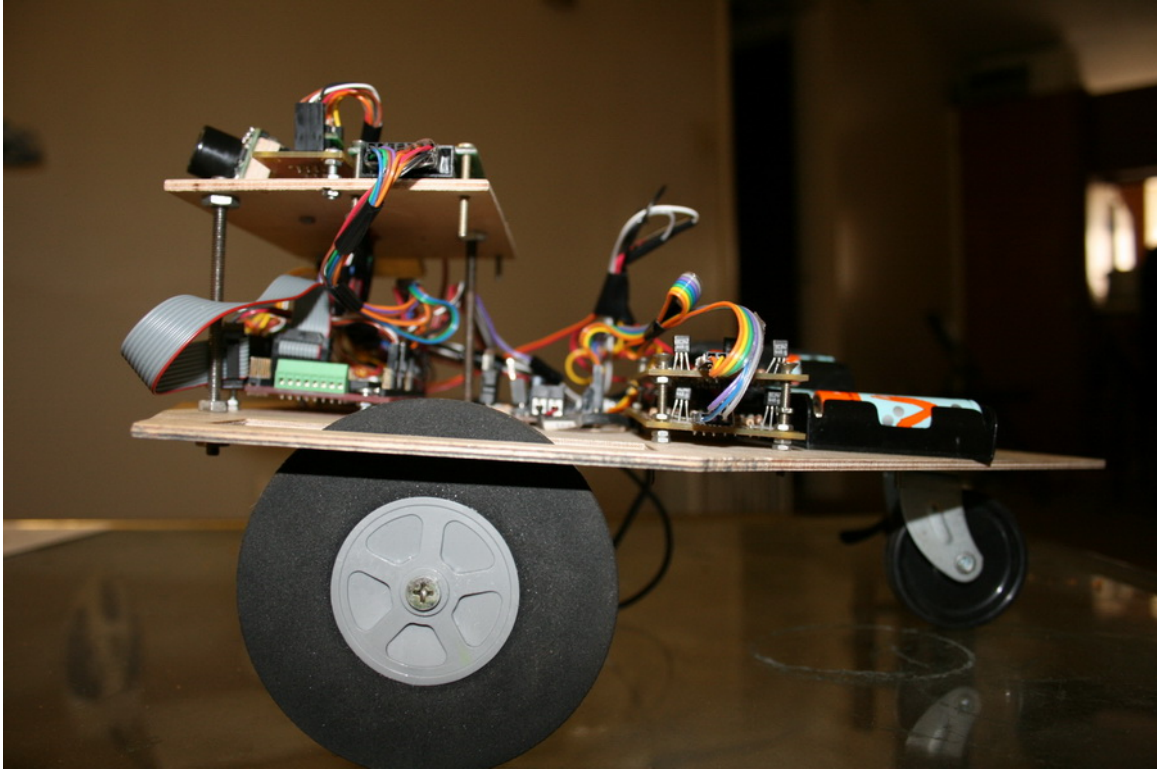


Autonomous NavigaTor (ANT)

a GPS based autonomous vehicle



Tarandeep Brar

UFID: 30113262

University of Florida
Department of Electrical and Computer Engineering
EEL 5666

Intelligent Machines Design Laboratory

Instructors: Dr. A. Antonio Arroyo

Dr. Eric M. Schwartz

TA: Mike Pridgen

Adam Barnett

Sara Keen

Table of contents

1. Cover Page.....	1
2. Table of Contents.....	2
3. Abstract.....	3
4. Executive Summary.....	4
5. Introduction.....	5
6. Integrated System.....	6
7. Mobile Platform.....	7
8. Actuation.....	9
9. Sensors.....	10
10. Behaviors.....	17
11. Experimental Layout and Results.....	18
12. Conclusion.....	19
13. Documentation.....	20
14. Appendices.....	15

Abstract

Autonomous NavigaTor or ANT as the name suggests is a self driven vehicle. It is designed to move from one location to other guided by variety of sensors. These include GPS, Sonar and Bump sensors. The short term objective of this project is to move from one place to another but later on as its range and accuracy is increased it can be used to guide people from one place to another or on larger scale it can be used in creating driverless cars.

The objectives of the project are as follows:

1. Build a platform which is rugged enough to move over varied terrain such as roads, grass.
2. Integrate sonar and bump sensors with the platform so that ANT can exhibit basic obstacle avoidance. This behavior will provide foundation to the final goal of the project.
3. Integrate GPS sensor to the robot so that it is cognizant of its current location and destination so as to reach its target using the best available path.
4. Build a platform which is rugged enough to move over varied terrain such as roads, grass.
5. Integrate sonar and bump sensors with the platform so that ANT can exhibit basic obstacle avoidance. This behavior will provide foundation to the final goal of the project.
6. Integrate GPS sensor to the robot so that it is cognizant of its current location and destination so as to reach its target using the best available path.

Executive Summary

ANT is a GPS based autonomous vehicle which moves from origin to destination through a number of waypoints avoiding obstacles. It is an ambitious project as everyone working on this architecture has got limited success, reason being inherent inaccuracy of GPS system and noise/interference while using obstacle avoidance sensors. Normal accuracy rate of GPS is around 10 meters. The sensors and microcontroller is mounted on custom designed 3 wheel platform with two front wheels providing drive and steering and third castor wheel balances the platform.

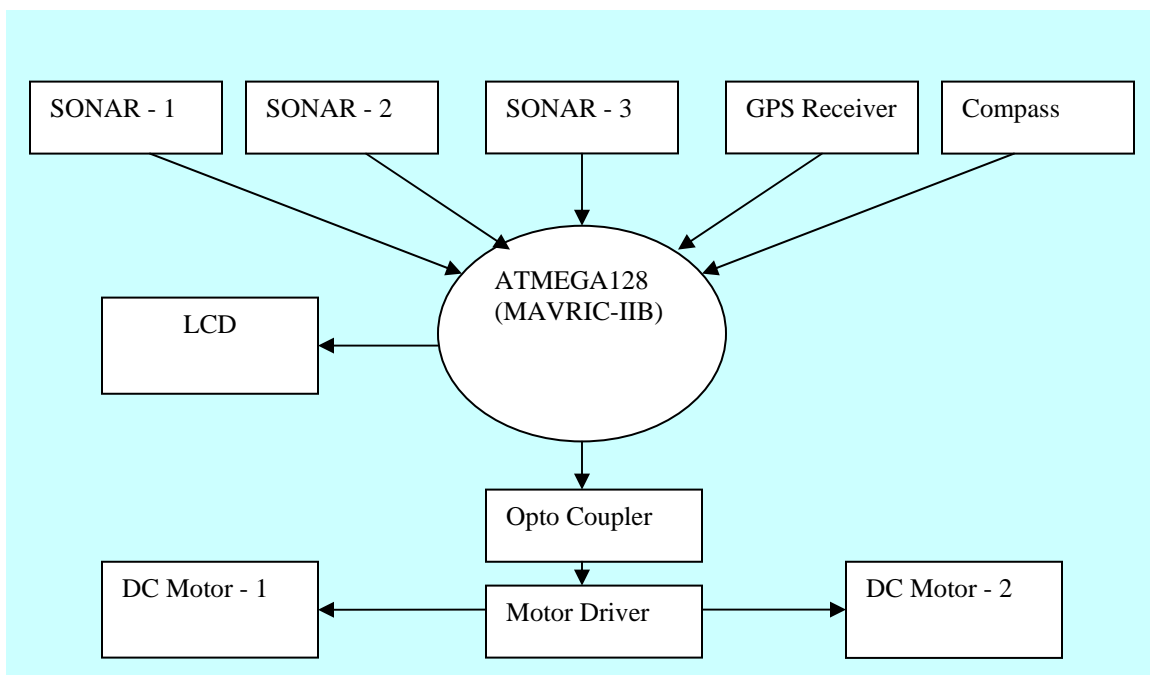
Before moving ANT is connected to a computer from where directions to its destination are fed. The instructions are in the form of GPS co-ordinates of its destination and the path which it will follow for reaching its destination. These directions can be received from Google maps or similar tool. Then ANT is switched to drive mode and it follows the path to its destination. When it encounters an obstacle in its path, ANT will slow down and scan the area immediate to where obstacle is detected. ANT will then steer left/right depending upon the situation to bypass the obstacle and reach its target.

Introduction

The main focus of this project is to develop a robot which can autonomously travel from one place to another, avoiding obstacles. To handle navigation, Global Positioning System (GPS) will be used. GPS is a system of 24 satellites which transmit position signals to earth. A GPS receiver uses data from 3 to 12 of these satellites to figure its exact location. Due to atmospheric conditions and other factors, the error in data received from GPS sensor has an error of 10 meters. Maxbotix ultrasonic sensors are used for obstacle avoidance. They have range of 4 meters i.e. they can detect an object from a distance of 4 meters. I am using 3 sensors to detect obstacles on left/right/center. LCD display will show errors, the path to follow and other information. Orientation information and direction information is provided by HM55b compass. The equipment is mounted on 3 wheel platform which is driven by 2 DC motors.

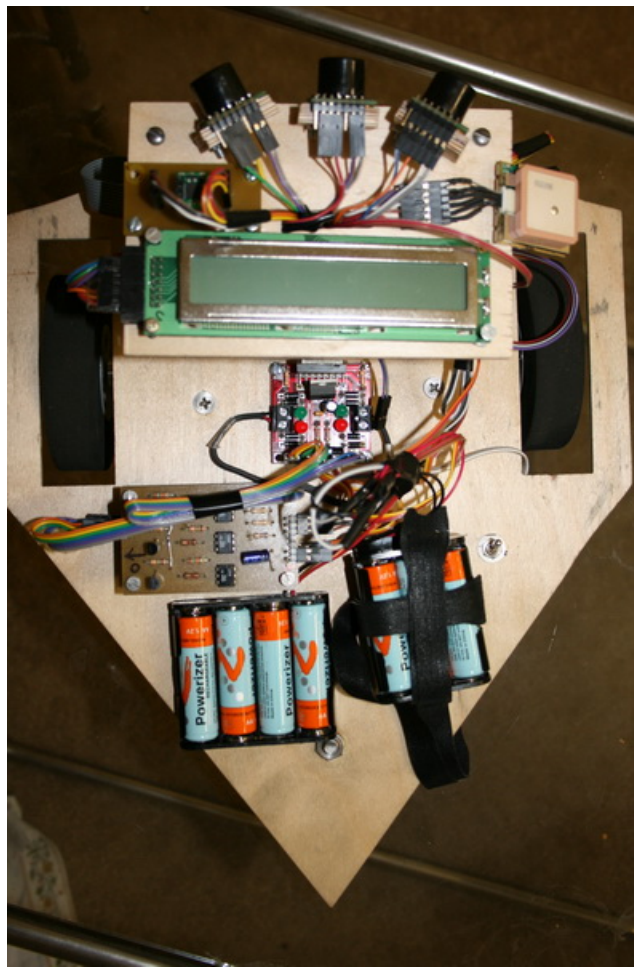
Integrated System

ANT is fitted with a GPS sensor (20 Channel EM-406A SiRF III Receiver), 1 HM55B compass and 3 Maxbotix sonar sensors. This is driven by platform mounted on two DC motors. LCD displays the decisions being taken by ANT. The brain of ANT is a MAVRIK-II B. MAVRIC-IIB is a powerful microcontroller board based on the ATmega128 MCU. It is fully programmable using familiar languages such as C and BASIC. The DC motors are driven by L298 motor driver which is connected to microcontroller using optocouplers so as to shield the microcontroller from spikes generated by motor. The microcontroller generates PWM signal to control the speed of motor.

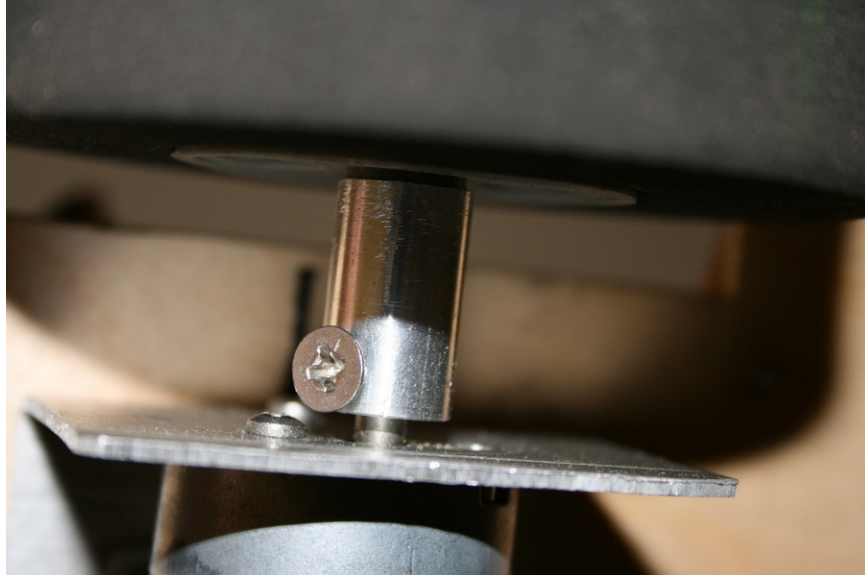


Mobile Platform

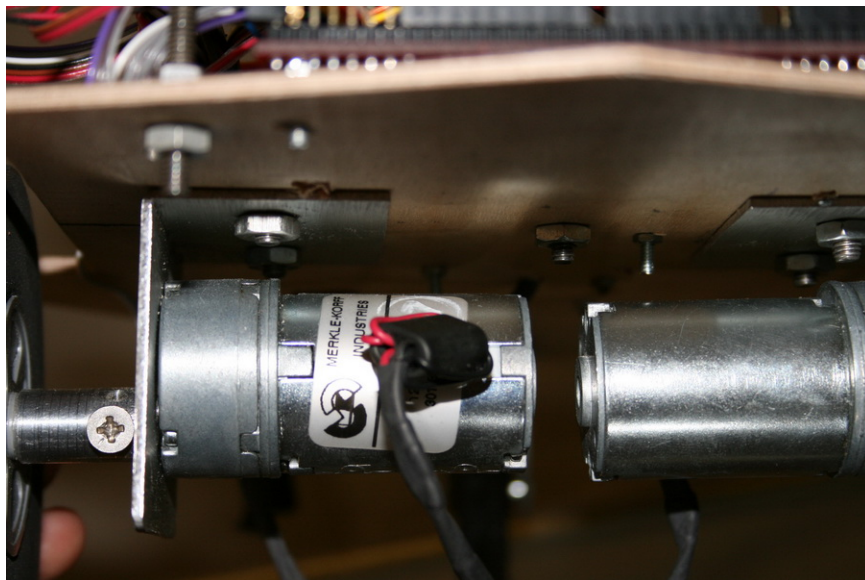
ANT is a three wheel robot with front wheel drive. It is driven using two powerful DC motors. Steering is done by front two wheels. I am planning to use custom built platform for running ANT. The platform is made out of strong wood. The motors are connected to wheels using custom built aluminum hubs and motors are connected to platform using custom built aluminum clamps. This provides a robust design. The batteries are placed in an easily accessible position. The 3' wheels bought from hobby shop provide adequate traction and road clearance to run ANT over all kinds of terrain.



Platform



Hub



Clamp

Actuation

ANT is driven by two Merkle-Korff gear reduction motors by Merkle-Korff industries. Merkle-Korff gear reduction motor is heavy duty motor. It runs at around 240RPM at 12Vdc and 120RPM at 6Vdc



Sensors

ANT shall navigate using variety of sensors which are as follows:

1. **GPS:** GPS Sensor will provide ANT its current position relative from which it can navigate to its destination. The GPS sensor used is EM-406A by GLOBALSAT.
2. **Sonar:** Sonar Sensor will be used by ANT to navigate between objects. 3 sonars mounted on front, right and left of ANT will provide the location of obstacles so that ANT can go between them without hitting them. The sonar sensor being used is LV-MaxSonar-EZ0 by MAXBOTIX.
3. **Compass:** Compass will be used to determine orientation of ANT with respect to its destination. The compass sensor used is HM55B compass module by HITACHI.

GPS Sensor:

The EM-406A GPS module from USGlobalSat is based on SiRF StarIII chipset. This complete module includes on-board voltage regulation, LED status indicator, battery backed RAM, and a built-in patch antenna.



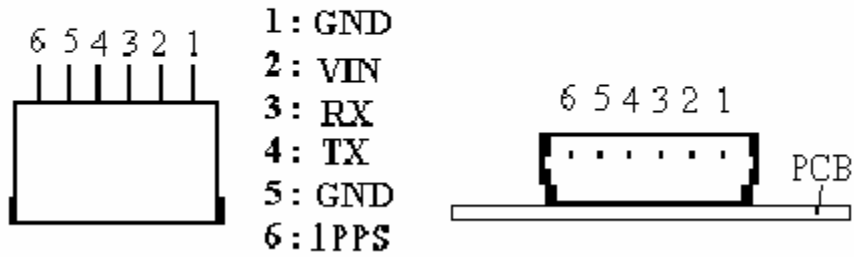
EM-406A SiRF III Receiver

Features:

- SiRF starIII high performance GPS Chip Set
- Very high sensitivity (Tracking Sensitivity: -159 dBm)
- Extremely fast TTFF (Time To First Fix) at low signal level
- Support NMEA 0183 data protocol
- Built-in SuperCap to reserve system data for rapid satellite acquisition
- Built-in patch antenna
- LED indicator for GPS fix or not fix

- Accuracy: 10 meters (2D RMS), 5 meters, (WAAS enabled)
- Protocol: TTL level (Output voltage level: 0V ~ 2.85V), RS-232 level
- Baud rate: 4,800 bps

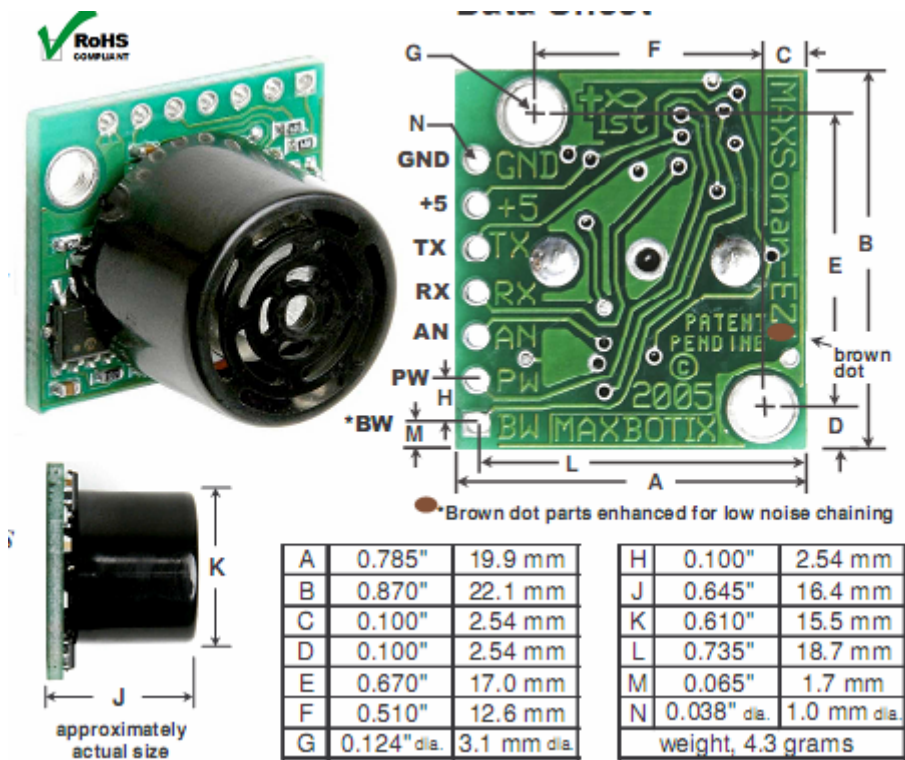
Pin Assignment



EM-406A SiRF III Receiver

MaxBotix Sonar

Since the robot will be running outdoors, light levels will be too high for IR to be of any use. Also, the useful distance of IR is less than 2 feet, which would be inadequate for the speeds at which the robot will be traveling. So, sonar has been chosen for object detection. The LV-MaxSonar-EZ1 detects objects from 0-inches to 254-inches (6.45-meters) and provides sonar range information from 6-inches out to 254-inches with 1-inch resolution. Objects from 0-inches to 6-inches range as 6-inches. The interface output formats included are pulse width output, analog voltage output, and serial digital output.



Features

- Continuously variable gain for beam control and side lobe suppression
- Object detection includes zero range objects
- 2.5V to 5.5V supply with 2mA typical current draw
- Readings can occur up to every 50mS, (20-Hz rate)
- Free run operation can continually measure and output range information
- Triggered operation provides the range reading as desired
- All interfaces are active simultaneously
 - Serial, 0 to Vcc
 - 9600Baud, 81N
 - Analog, (Vcc/512) / inch
 - Pulse width, (147uS/inch)
- Learns ringdown pattern when commanded to start ranging
- Designed for protected indoor environments
- Sensor operates at 42KHz
- High output square wave sensor drive (double Vcc)

Benefits

- Very low cost sonar ranger
- Reliable and stable range data
- Sensor dead zone virtually gone
- Lowest power ranger
- Quality beam characteristics
- Mounting holes provided on the circuit board
- Very low power ranger, excellent for multiple sensor or battery based systems
- Can be triggered externally or internally
- Sensor reports the range reading directly, frees up user processor
- Fast measurement cycle
- User can choose any of the three sensor outputs

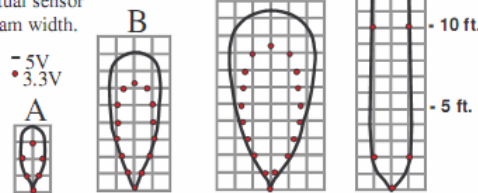
Beam Characteristics

People detection requires high sensitivity, yet a narrow beam angle requires low sensitivity. The LV-MaxSonar®-EZ1™ balances the detection of people with a narrow beam width.

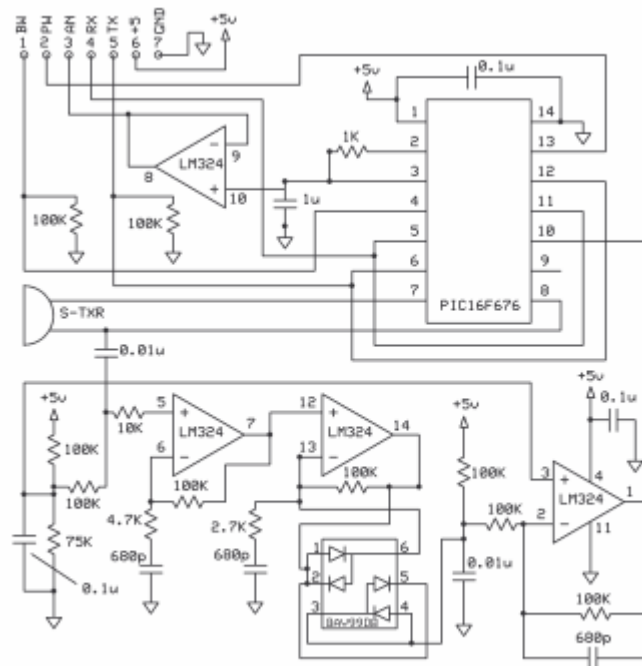
Sample results for measured beam patterns are shown below on a 12-inch grid. The detection pattern is shown for;

- (A) 0.25-inch diameter dowel, note the narrow beam for close small objects,
- (B) 1-inch diameter dowel, note the long narrow detection pattern,
- (C) 3.25-inch diameter rod, note the long controlled detection pattern,
- (D) 11-inch wide board moved left to right with the board parallel to the front sensor face and the sensor stationary. This shows the sensor's range capability.

Note: The displayed beam width of (D) is a function of the specular nature of sonar and the shape of the board (i.e. flat mirror like) and should never be confused with actual sensor beam width.

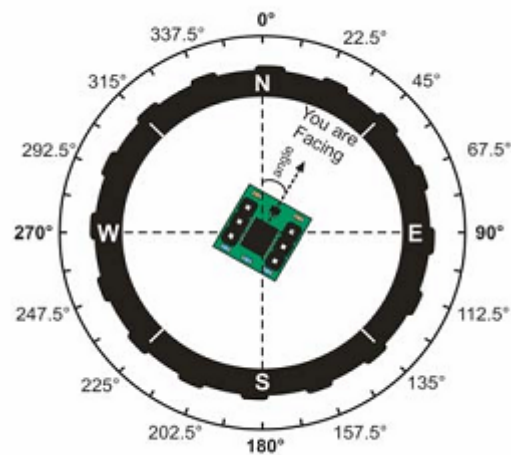
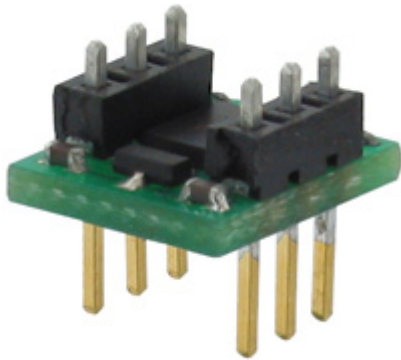


beam characteristics are approximate



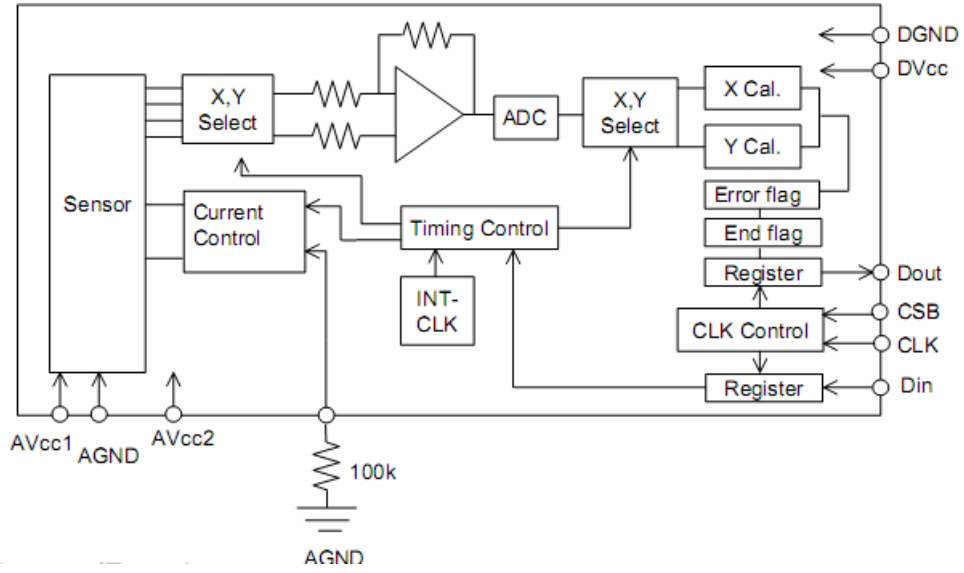
Compass

Hitachi HM55B Compass Module is a dual-axis magnetic field sensor. The compass module provides measurements with a synchronous serial interface. It is capable of detecting microtesla variations in magnetic field strength. North/south and east/west sensing axes make it easy for microcontroller to resolve and display direction in a 360 deg format with magnetic north at 0 deg, northeast at 45 deg, east at 90 deg, south at 180 deg and so on.



Features

- Sensitive to microtesla (μT) variations in magnetic field strength
- Simplifies direction by resolving magnetic field measurements into two component axes
- Good for 6-bit (64-direction) resolution measurements after software calibration
- Only 30 to 40 ms between start measurement and data-ready
- Built-in resistor protection for data pins eliminates bus conflict risks
- Compact and breadboard-friendly 0.3 inch, 6-pin DIP package



Behavior

ANT begins by determining its target coordinates and path to follow from computer. Then it follows the destined route to reach its target. Once it is moving the sonar constantly checks for obstacles in its way. If there is something within 5 to 15 feet, ANT will slow down. If there is something within 3 feet in front of ANT, it will stop, scan the vicinity and determine the best route around the obstacle. ANT stores the co-ordinates of the path which it needs to follow on onboard memory and it compares its current position to calculate the route it needs to take to reach destination.

Experimental Layout and Results

All experiments were conducted with a fully operational platform. The sonar worked perfectly until it was installed on the car. The height of the transducer caused premature echo returns. This was probably due to the proximity of the ground. Also as I was using multiple sonars there was lot of noise from other sonars. Averaging out the data also didn't help much. So I had disable/ enable the sonars one at a time and get measurement. This resulted in less noisy data. The vehicle must also operate on a smooth surface or the sonar will give false readings. The compass never performed as expected. I tried different software which was working with same model of compass but it didn't work on my compass. So I think this was hardware issue and not a software issue. GPS provided which was quite noisy. I think this is due to buffering provided by avr-lib.

Conclusion

The most significant problem with this project was the compass. For some reason I could not get data from the compass. I'm still trying to determine whether it is my software or the compass hardware that is the problem. The ability to get a current heading is critical for producing any algorithm, which controls the car's movements. I can get a heading from the GPS unit using the (\$PASHS,NME,POS) command. I have reservations about the practicality of using this data because there is a certain amount of error when calculating the coordinates. I could get by without this data, but tends to make the platform's motion jerky. The sonar and GPS worked perfectly for the most part. I thought the GPS would be the most difficult part of this project, instead it was the easiest to implement. The work on ANT is not finished and will take some time before its working correctly.

Source Code:

Ant.c

```
#include <avr/io.h> // include I/O definitions (port names,
pin names, etc)
#include "lcd.h"
#include "timer.h"
#include "rprintf.h"
#include "Sonar.h"
#include "Global.h"
#include "Motor.h"
#include "uart2.h"
#include "gps.h"
#include "nmea.h"

#define GO_LEFT_SLOW Motor_SetSpeed(MOTOR_LEFT,
MOTOR_SPEED_STOP); Motor_SetSpeed(MOTOR_RIGHT, MOTOR_SPEED_SLOW);
#define GO_RIGHT_SLOW Motor_SetSpeed(MOTOR_RIGHT,
MOTOR_SPEED_STOP); Motor_SetSpeed(MOTOR_LEFT, MOTOR_SPEED_SLOW);
#define GO_LEFT_FAST Motor_SetSpeed(MOTOR_LEFT,
MOTOR_SPEED_STOP); Motor_SetSpeed(MOTOR_RIGHT, MOTOR_SPEED_NORMAL);
// #define GO_LEFT_FAST Motor_SetSpeed(MOTOR_LEFT,
MOTOR_SPEED_SLOW); Motor_SetSpeed(MOTOR_RIGHT, MOTOR_SPEED_NORMAL);
#define GO_RIGHT_FAST Motor_SetSpeed(MOTOR_RIGHT,
MOTOR_SPEED_STOP); Motor_SetSpeed(MOTOR_LEFT, MOTOR_SPEED_NORMAL);
// #define GO_RIGHT_FAST Motor_SetSpeed(MOTOR_RIGHT,
MOTOR_SPEED_SLOW); Motor_SetSpeed(MOTOR_LEFT, MOTOR_SPEED_NORMAL);
#define GO_SLOW Motor_SetSpeed(MOTOR_RIGHT,
MOTOR_SPEED_SLOW); Motor_SetSpeed(MOTOR_LEFT, MOTOR_SPEED_SLOW);
#define GO_NORMAL Motor_SetSpeed(MOTOR_RIGHT,
MOTOR_SPEED_NORMAL); Motor_SetSpeed(MOTOR_LEFT, MOTOR_SPEED_NORMAL);
#define GO_FAST Motor_SetSpeed(MOTOR_RIGHT,
MOTOR_SPEED_FAST); Motor_SetSpeed(MOTOR_LEFT, MOTOR_SPEED_FAST);
#define GO_REVERSE Motor_SetDirection(MOTOR_DIR_REV);
GO_SLOW;
#define GO_FWD Motor_SetDirection(MOTOR_DIR_FWD);
GO_SLOW;

extern unsigned short uartRxOverflow[2];

int sonarLeft[AVG_ARRAY_SIZE];
int sonarCenter[AVG_ARRAY_SIZE];
int sonarRight[AVG_ARRAY_SIZE];
int nSonarIndex = 0; //store the current pos in the global array

float latitude[WAYPOINT_SIZE];
float longitude[WAYPOINT_SIZE];
int direction[WAYPOINT_SIZE];
float curLat;
float curLong;

void DataClear(void);
int AvgSonarDist(int nSonar);
void UpdateSonarData(void);
void InitWaypoints(void);
```

```
int main(void)
{
    timerInit();           //Initialize timer

    lcdInit();            //Initialize LCD
LCD    rprintfInit(lcdDataWrite);    //Redirect rprintf output to

    Sonar_Init();        // Initialize sonar ports

    Motor_Init();        // Initialize Motor ports

    uart1Init();         // Initialize serial port1
    nmeaInit();          // Initialize NMEA
    gpsInit();           // Initialize GPS(currently does
nothing)
    uartSetBaudRate(1, 4800);    // set baud rate

    InitWaypoints();     // Initialize the waypoint datastructure
    int nextWaypt = 0;

    bit_clear(TOGGLE_SWTCH_DATA_DDR, BIT(TOGGLE_SWTCH_PIN));
    // Activate the toggle switch port
    // make portC 0th bit as input pin.
    bit_set(TOGGLE_SWTCH_DATA_PORT, BIT(TOGGLE_SWTCH_PIN));

    GO_FWD;
//    rprintfProgStrM("Push Button to move");
//while((PINF & _BV(0)) >> (0))
//{
//    timerPause(500);
//}
//timerPause(500);

//    GO_STRAIGHT_NORMAL;
float tempLat = 0.00;
float tempLong = 0.00;
float lastLatDiff = 0.10; // to calculate the distance we have
travelled
float lastLongDiff = 0.10; // to calculate the distance we have
travelled

while(1/(PINF & _BV(0)) >> (0)*/)
{
    lcdClear();
    //UpdateSonarData();
    //int nDist = AvgSonarDist(SONAR_CENTER);
    int nDist = Sonar_GetDist(SONAR_CENTER);
    rprintfNum(10,4,0,' ',Sonar_GetDist(SONAR_LEFT));
    //rprintfNum(10,4,0,' ',AvgSonarDist(SONAR_LEFT));
    //rprintfNum(10,4,0,' ',Sonar_GetDist(SONAR_CENTER));
    rprintfNum(10,4,0,' ', nDist);
    rprintfNum(10,4,0,' ',Sonar_GetDist(SONAR_RIGHT));
    //timerPause(TIME_100_MSEC);
    //continue;
    //rprintfNum(10,4,0,' ',AvgSonarDist(SONAR_RIGHT));
    while( nmeaProcess(uartGetRxBuffer(1)) );
}
```

```

        //lcdGotoXY(0, 0);
        tempLat = 180*(gpsGetInfo()->PosLLA.lat.f/PI);
        tempLong = 180*(gpsGetInfo()->PosLLA.lon.f/PI);
        // if the gps data is within the threshold update current
positions
        if ((tempLat > GPS_LAT_MIN) && (tempLat < GPS_LAT_MAX) &&
(tempLong > GPS_LONG_MIN) && (tempLong < GPS_LONG_MAX))
        {
            curLat = tempLat;
            curLong = tempLong;
        }
        tempLat = latitude[nextWaypt] - curLat;
        tempLong = longitude[nextWaypt] - curLong;
        if (tempLat > lastLatDiff && tempLong < lastLongDiff)
        {
            // we are going offcourse in Latitude
        }
        else if (tempLat < lastLatDiff && tempLong > lastLongDiff)
        {
            // we are going offcourse in Longitude
        }
        else if (tempLat > lastLatDiff && tempLong > lastLongDiff)
        {
            // we are completely offcourse
        }
        else if (tempLat < lastLatDiff && tempLong < lastLongDiff)
        {
            // success we are on course
        }

        lastLongDiff = longitude[nextWaypt] - curLong;
        lastLatDiff = latitude[nextWaypt] - curLat;

        //rprintfProgStrM("Lat:");    rprintfFloat(15,
180*(gpsGetInfo()->PosLLA.lat.f/PI));
        //lcdGotoXY(0, 1);
        //rprintfProgStrM("");*/    rprintfFloat(15,
180*(gpsGetInfo()->PosLLA.lon.f/PI));
        //rprintf(",%d", uartRxOverflow[1]);
        if (nDist < DIST_MAX_TURNRADII && nDist > DIST_MIN_CENTER)
        {
            GO_SLOW;
        }
        else if (nDist < DIST_MIN_CENTER) //Obstacle detected
        {
            //if (AvgSonarDist(SONAR_LEFT) > DIST_MAX_TURNRADII)
            if (Sonar_GetDist(SONAR_LEFT) > DIST_MAX_TURNRADII)
            {
                lcdGotoXY(0, 1);
                rprintf("Turning Left");
                //UpdateSonarData();
                GO_LEFT_FAST;
                //while (AvgSonarDist(SONAR_CENTER) <
DIST_MAX_TURNRADII)
                while (Sonar_GetDist(SONAR_CENTER) <
DIST_MAX_TURNRADII || Sonar_GetDist(SONAR_RIGHT) < DIST_MIN_RIGHT)
            {

```

```

        //UpdateSonarData();
        timerPause(TIME_1_SEC);
    }
    GO_NORMAL;
    lcdGotoXY(0, 1);
    rprintf("Going Straight");
}
//else if (AvgSonarDist(SONAR_RIGHT) >
DIST_MAX_TURNRADII)
else if (Sonar_GetDist(SONAR_RIGHT) >
DIST_MAX_TURNRADII)
{
    lcdGotoXY(0, 1);
    rprintf("Turning Right");
    //UpdateSonarData();
    GO_RIGHT_FAST;
    //while (AvgSonarDist(SONAR_CENTER) <
DIST_MAX_TURNRADII)
        while (Sonar_GetDist(SONAR_CENTER) <
DIST_MAX_TURNRADII || Sonar_GetDist(SONAR_LEFT) < DIST_MIN_LEFT)
        {
            //UpdateSonarData();
            timerPause(TIME_1_SEC);
        }
        GO_NORMAL;
        lcdGotoXY(0, 1);
        rprintf("Going Straight");
    }
else
{
    GO_REVERSE;
    //while (AvgSonarDist(SONAR_CENTER) <
DIST_MAX_TURNRADII)
        while (Sonar_GetDist(SONAR_CENTER) <
DIST_MAX_TURNRADII)
        {
            //UpdateSonarData();
            timerPause(TIME_1_SEC);
        }
        Motor_SetDirection(MOTOR_DIR_FWD);
        GO_LEFT_FAST;
        timerPause(TIME_5_SEC);
        GO_NORMAL;
    }

}
//UpdateSonarData();
// rprintf("Distance :");

//    uartFlushReceiveBuffer(1);
//    timerPause(TIME_100_MSEC);
//    timerPause(100);
}
return 0;
}

void UpdateSonarData(void)

```

```
{
    if (nSonarIndex >= AVG_ARRAY_SIZE)
        nSonarIndex = 0; //reset index
    sonarLeft[nSonarIndex] = Sonar_GetDist(SONAR_LEFT);
    sonarCenter[nSonarIndex] = Sonar_GetDist(SONAR_CENTER);
    sonarRight[nSonarIndex] = Sonar_GetDist(SONAR_RIGHT);

    nSonarIndex++;
}

void DataClear(void)
{
    int nIndex = 0;
    while (nIndex++ < AVG_ARRAY_SIZE)
    {
        sonarLeft[nIndex] = 0;
        sonarCenter[nIndex] = 0;
        sonarRight[nIndex] = 0;
    }
}

int AvgSonarDist(int nSonar)
{
    int nIndex = 0;
    int nDist = 0;
    int nTotal = 0;
    while (nIndex++ < AVG_ARRAY_SIZE)
    {
        int nTemp = 0;

        if (nSonar == SONAR_LEFT)
            nTemp += sonarLeft[nIndex];
        else if (nSonar == SONAR_CENTER)
            nTemp += sonarCenter[nIndex];
        else if (nSonar == SONAR_RIGHT)
            nTemp += sonarRight[nIndex];

        if (nTemp > 4 && nTemp < 200)
        {
            nDist += nTemp;
            nTotal++;
        }
    }
    if (nTotal > 0) // to prevent divide by zero;
        return nDist/nTotal;
    return 0;
}

void InitWaypoints(void)
{
    latitude[0] = 0.0;
    longitude[0] = 0.0;
    direction[0] = 0.0;

    latitude[1] = 0.0;
    longitude[1] = 0.0;
}
```

```
direction[1] = 0.0;

latitude[2] = 0.0;
longitude[2] = 0.0;
direction[2] = 0.0;

latitude[3] = 0.0;
longitude[3] = 0.0;
direction[3] = 0.0;

latitude[4] = 0.0;
longitude[4] = 0.0;
direction[4] = 0.0;
}
```

Global.h

```
#ifndef __GLOBAL_H__
#define __GLOBAL_H__

// global AVRLIB defines
#include "avrlibdefs.h"
// global AVRLIB types definitions
#include "avrlibtypes.h"

#define RPRINTF_FLOAT
#define CYCLES_PER_US ((F_CPU+500000)/1000000) // cpu cycles per
microsecond

#define AVG_ARRAY_SIZE          3

// GPS threshold to weed out sensor noise
#define GPS_LAT_MIN             28.50
#define GPS_LAT_MAX             30.00
#define GPS_LONG_MIN           -82.25
#define GPS_LONG_MAX           -82.40

// Obstacle avoidance threshold
#define DIST_MIN_CENTER         16
#define DIST_MIN_LEFT          12
#define DIST_MIN_RIGHT         12
#define DIST_MAX_TURNRADIUS    24

// Delay times
#define TIME_10_MSEC           10
#define TIME_100_MSEC          100
#define TIME_1_SEC              1000
#define TIME_5_SEC              5000

// Toggle switch
#define TOGGLE_SWTCH_DATA_DDR   DDRF
#define TOGGLE_SWTCH_DATA_PORT  PORTF
#define TOGGLE_SWTCH_DATA_PIN   PINF
#define TOGGLE_SWTCH_PIN        0

// Sonar details
#define SONAR_DATA_DDR          DDRA
#define SONAR_DATA_PORT         PORTA
#define SONAR_DATA_PIN          PINA
#define SONAR_1_ECHOPW_PIN      0
#define SONAR_2_ECHOPW_PIN      1
#define SONAR_3_ECHOPW_PIN      2
#define SONAR_1_TRIGGERRX_PIN    3
#define SONAR_2_TRIGGERRX_PIN    4
#define SONAR_3_TRIGGERRX_PIN    5

#define SONAR_LEFT              1
#define SONAR_CENTER            2
#define SONAR_RIGHT             3

//Motor details
#define MOTOR_DATA_DDR          DDRB
```

```
#define MOTOR_DATA_PORT      PORTB
#define MOTOR_DATA_PIN      PINB
#define MOTOR_DIR_DDR       DDRE
#define MOTOR_DIR_PORT      PORTE
#define MOTOR_DIR_PIN       PINE

#define MOTOR_RIGHT_EN_PIN  6
#define MOTOR_RIGHT_RV_PIN  1
#define MOTOR_RIGHT_FW_PIN  2
#define MOTOR_LEFT_EN_PIN   5
#define MOTOR_LEFT_FW_PIN   5
#define MOTOR_LEFT_RV_PIN   4

#define MOTOR_LEFT          1
#define MOTOR_RIGHT         2

// Direction settings
#define MOTOR_DIR_FWD       1
#define MOTOR_DIR_REV       2

#define MOTOR_RIGHT_OCR     OCR1A
#define MOTOR_LEFT_OCR      OCR1B

#define MOTOR_SPEED_STOP    0
#define MOTOR_SPEED_SLOW    5000
#define MOTOR_SPEED_NORMAL  10000
#define MOTOR_SPEED_FAST    20000

// Compass Details
#define COMPASS_DATA_DDR    DDRE
#define COMPASS_DATA_PORT   PORTE
#define COMPASS_DATA_PIN    PINE

#define COMPASS_DIN         4
#define COMPASS_DOUT        5
#define COMPASS_EN          6
#define COMPASS_CLK         7
#define CLOCK_TICKS        100

unsigned int clkTicks;
#define DELAY(TICKS) clkTicks = 0; while(clkTicks<TICKS)clkTicks++;
#define DELAY_LONG
    DELAY(65000);DELAY(65000);DELAY(65000);DELAY(65000);DELAY(65000);

// Bit operations
#define bit_get(p,m) ((p) & (m))
#define bit_set(p,m) ((p) |= (m))
#define bit_clear(p,m) ((p) &= ~(m))
#define bit_flip(p,m) ((p) ^= (m))
#define bit_write(c,p,m) (c ? bit_set(p,m) : bit_clear(p,m))
#define BIT(x) (0x01 << (x))
#define LONGBIT(x) ((unsigned long)0x00000001 << (x))

#define WAYPOINT_SIZE 5
#define DIR_STRT 0
#define DIR_LEFT 1
#define DIR_RIGHT 2
```

```
#endif // #ifndef __GLOBAL_H__
```

Motor.h

```
#ifndef __MOTOR_H__
#define __MOTOR_H__

void Motor_Init(void);
void Motor_SetSpeed(int Motor, int nSpeed);
void Motor_Stop(void);
void Motor_SetDirection(int Direction);
//void Motor_Forward(void);

#endif
```

Motor.c

```
#include <avr/io.h>
#include "Motor.h"
#include "Global.h"

unsigned int LeftMotorSpeed = 0;
unsigned int RightMotorSpeed = 0;

void Motor_Init(void)
{
    TCCR1A = 0b10100000;    // .....00:P&FC PWM //
11.....,..11.....:OC1A,OC1B inv
    TCCR1B = 0b00010010;    // ...10....:P&FC PWM // .....010:bclk/8
(~8kHz)

    bit_set(MOTOR_DATA_DDR, BIT(MOTOR_RIGHT_EN_PIN));
    bit_set(MOTOR_DATA_DDR, BIT(MOTOR_LEFT_EN_PIN));

    bit_set(MOTOR_DATA_PORT, BIT(MOTOR_RIGHT_EN_PIN));
    bit_set(MOTOR_DATA_PORT, BIT(MOTOR_LEFT_EN_PIN));

    bit_set(MOTOR_DIR_DDR, BIT(MOTOR_RIGHT_FW_PIN));
    bit_set(MOTOR_DIR_DDR, BIT(MOTOR_LEFT_FW_PIN));
    bit_set(MOTOR_DIR_DDR, BIT(MOTOR_RIGHT_RV_PIN));
    bit_set(MOTOR_DIR_DDR, BIT(MOTOR_LEFT_RV_PIN));

    ICR1=20000;
    TCNT1=0x0000;
    MOTOR_RIGHT_OCR = 0x0000;           // 1/2 duty, ~2.5v dc
level, motor 1 on PB5 (OC1A)
    MOTOR_LEFT_OCR = 0x0000;           // 0/256 duty, ~2.5v dc
level, motor 2 on PB6 (OC1B)
}

void Motor_SetSpeed(int Motor, int nSpeed)
{
    int curSpeed = 0;
    if (Motor == MOTOR_LEFT)
        curSpeed = LeftMotorSpeed;
    else if (Motor == MOTOR_RIGHT)
        curSpeed = RightMotorSpeed;
    if (nSpeed > curSpeed)
    {
        while (curSpeed < nSpeed)
        {
            curSpeed += 100;

            if (Motor == MOTOR_LEFT)
            {
                MOTOR_LEFT_OCR = curSpeed;
                LeftMotorSpeed = curSpeed;
            }
            else if (Motor == MOTOR_RIGHT)
            {
                MOTOR_RIGHT_OCR = curSpeed;
            }
        }
    }
}
```

```
        RightMotorSpeed = curSpeed;
    }
    DELAY(500);
}
}
else if (nSpeed < curSpeed)
{
    while (curSpeed > nSpeed)
    {
        curSpeed -= 100;

        if (Motor == MOTOR_LEFT)
        {
            MOTOR_LEFT_OCR = curSpeed;
            LeftMotorSpeed = curSpeed;
        }
        else if (Motor == MOTOR_RIGHT)
        {
            MOTOR_RIGHT_OCR = curSpeed;
            RightMotorSpeed = curSpeed;
        }
        DELAY(500);
    }
}
}
/*
void Motor_Slow(int Motor, int nIncrements)
{
}
*/
void Motor_Stop(void)
{
    bit_clear(MOTOR_DIR_PORT, BIT(MOTOR_RIGHT_RV_PIN));
    bit_clear(MOTOR_DIR_PORT, BIT(MOTOR_LEFT_RV_PIN));
    bit_clear(MOTOR_DIR_PORT, BIT(MOTOR_RIGHT_FW_PIN));
    bit_clear(MOTOR_DIR_PORT, BIT(MOTOR_LEFT_FW_PIN));

    MOTOR_LEFT_OCR = MOTOR_SPEED_STOP;
    MOTOR_RIGHT_OCR = MOTOR_SPEED_STOP;

    LeftMotorSpeed = 0;
    RightMotorSpeed = 0;
    DELAY(1000);
}

void Motor_SetDirection(int Direction)
{
    Motor_Stop();
    if (Direction == MOTOR_DIR_REV)
    {
        bit_set(MOTOR_DIR_PORT, BIT(MOTOR_RIGHT_RV_PIN));
        bit_set(MOTOR_DIR_PORT, BIT(MOTOR_LEFT_RV_PIN));
    }
    else if (Direction == MOTOR_DIR_FWD)
    {
        bit_set(MOTOR_DIR_PORT, BIT(MOTOR_RIGHT_FW_PIN));
        bit_set(MOTOR_DIR_PORT, BIT(MOTOR_LEFT_FW_PIN));
    }
}
```

```
    }  
    DELAY(1000);  
}
```

Sonar.h

```
#ifndef __SUBSONAR_H__
#define __SUBSONAR_H__

/*-----
-----FUNCTIONS PROTOTYPES-----
-----*/

void Sonar_Init(void); // TO INITIALIZE PORTS FOR SONAR
int Sonar_GetDist(int nSonar); //TO TRIGGER THE SONAR AND READ OBSTACLE
DISTANCE

#endif
```

Sonar.c

```
/*
subsonar.c

this code doesnt have main() and hence cant run on its own.
this is designed to be called from the code that runs the main logic

*/

/*-----
-----HEADER FILES-----
-----*/
#include <avr/io.h>
#include "Sonar.h"
#include "Global.h"
#include <util/delay.h>
#include "timer.h"

/*-----
-----CONNECTION BETWEEN SONAR AND ATMEGA128-----
-----*/

/*-----
-----CONTROL BITS OF SONAR -----
-----*/

#define SONAR_SET_TRIGGERRX1 bit_set(SONAR_DATA_PORT,
BIT(SONAR_1_TRIGGERRX_PIN))//SONAR_DATA_PORT|=_BV(SONAR_1_TRIGGERRX_PIN
)
#define SONAR_CLEAR_TRIGGERRX1 bit_clear(SONAR_DATA_PORT,
BIT(SONAR_1_TRIGGERRX_PIN))//SONAR_DATA_PORT&=~_BV(SONAR_1_TRIGGERRX_PI
N)
#define SONAR_SET_TRIGGERRX2 bit_set(SONAR_DATA_PORT,
BIT(SONAR_2_TRIGGERRX_PIN))//SONAR_DATA_PORT|=_BV(SONAR_2_TRIGGERRX_PIN
)
#define SONAR_CLEAR_TRIGGERRX2 bit_clear(SONAR_DATA_PORT,
BIT(SONAR_2_TRIGGERRX_PIN))//SONAR_DATA_PORT&=~_BV(SONAR_2_TRIGGERRX_PI
N)
#define SONAR_SET_TRIGGERRX3 bit_set(SONAR_DATA_PORT,
BIT(SONAR_3_TRIGGERRX_PIN))//SONAR_DATA_PORT|=_BV(SONAR_3_TRIGGERRX_PIN
)
#define SONAR_CLEAR_TRIGGERRX3 bit_clear(SONAR_DATA_PORT,
BIT(SONAR_3_TRIGGERRX_PIN))//SONAR_DATA_PORT&=~_BV(SONAR_3_TRIGGERRX_PI
N)

/*-----
-----FUNCTION TO INITIALIZE PORTS FOR SONAR-----
-----*/
void Sonar_Init(void)
{
```

```

    bit_set(SONAR_DATA_DDR, BIT(SONAR_1_TRIGGERRX_PIN));
//    SONAR_DATA_DDR |= _BV(1<<SONAR_1_TRIGGERRX_PIN);
//    SONAR_DATA_DDR |= _BV(SONAR_TRIGGERRX_PIN);
//    // set bit0 OF DDR to make it an output pin for TriggerRX
    bit_clear(SONAR_DATA_PORT, BIT(SONAR_1_TRIGGERRX_PIN));
//    SONAR_DATA_PORT &= ~_BV(1<<SONAR_1_TRIGGERRX_PIN);
//    SONAR_DATA_PORT &=~_BV(SONAR_TRIGGERRX_PIN);
    bit_set(SONAR_DATA_DDR, BIT(SONAR_2_TRIGGERRX_PIN));
//    SONAR_DATA_DDR |= _BV(1<<SONAR_2_TRIGGERRX_PIN);
//    SONAR_DATA_DDR |= _BV(SONAR_TRIGGERRX_PIN);
//    // set bit0 OF DDR to make it an output pin for TriggerRX
    bit_clear(SONAR_DATA_PORT, BIT(SONAR_2_TRIGGERRX_PIN));
//    SONAR_DATA_PORT &= ~_BV(1<<SONAR_2_TRIGGERRX_PIN);
//    SONAR_DATA_PORT &=~_BV(SONAR_TRIGGERRX_PIN);
    bit_set(SONAR_DATA_DDR, BIT(SONAR_3_TRIGGERRX_PIN));
//    SONAR_DATA_DDR |= _BV(1<<SONAR_3_TRIGGERRX_PIN);
//    SONAR_DATA_DDR |= _BV(SONAR_TRIGGERRX_PIN);
//    // set bit0 OF DDR to make it an output pin for TriggerRX
    bit_clear(SONAR_DATA_PORT, BIT(SONAR_3_TRIGGERRX_PIN));
//    SONAR_DATA_PORT &= ~_BV(1<<SONAR_3_TRIGGERRX_PIN);
//    SONAR_DATA_PORT &=~_BV(SONAR_TRIGGERRX_PIN);
//    // intialize the output value on TriggerRX as LOW

//    SONAR_DATA_DDR &=~_BV(1<<SONAR_1_ECHOPW_PIN);
    bit_clear(SONAR_DATA_DDR, BIT(SONAR_1_ECHOPW_PIN));
//    SONAR_DATA_DDR &=~_BV(SONAR_1_ECHOPW_PIN);
//    //clear bit1 OF DDR to make it an input pin to read EchoPW
//    SONAR_DATA_PORT |= _BV(1<<SONAR_1_ECHOPW_PIN);
    bit_set(SONAR_DATA_PORT, BIT(SONAR_1_ECHOPW_PIN));
//    SONAR_DATA_PORT |= _BV(SONAR_1_ECHOPW_PIN);
//    // set to enable the pull-up resistor on the input pin to read
EchoPW
//    SONAR_DATA_DDR &=~_BV(1<<SONAR_2_ECHOPW_PIN);
    bit_clear(SONAR_DATA_DDR, BIT(SONAR_2_ECHOPW_PIN));
//    SONAR_DATA_DDR &=~_BV(SONAR_2_ECHOPW_PIN);
//    //clear bit1 OF DDR to make it an input pin to read EchoPW
//    SONAR_DATA_PORT |= _BV(1<<SONAR_2_ECHOPW_PIN);
    bit_set(SONAR_DATA_PORT, BIT(SONAR_2_ECHOPW_PIN));
//    SONAR_DATA_PORT |= _BV(SONAR_2_ECHOPW_PIN);
//    // set to enable the pull-up resistor on the input pin to read
EchoPW
//    SONAR_DATA_DDR &=~_BV(1<<SONAR_3_ECHOPW_PIN);
    bit_clear(SONAR_DATA_DDR, BIT(SONAR_3_ECHOPW_PIN));
//    SONAR_DATA_DDR &=~_BV(SONAR_3_ECHOPW_PIN);
//    //clear bit1 OF DDR to make it an input pin to read EchoPW
//    SONAR_DATA_PORT |= _BV(1<<SONAR_3_ECHOPW_PIN);
    bit_set(SONAR_DATA_PORT, BIT(SONAR_3_ECHOPW_PIN));
//    SONAR_DATA_PORT |= _BV(SONAR_3_ECHOPW_PIN);
//    // set to enable the pull-up resistor on the input pin to read
EchoPW
}

/*-----FUNCTION TO TRIGGER THE SONAR AND READ OBSTACLE DISTANCE-----*/

```

```
int Sonar_GetDist(int nSonar)
{
    timerPause(50);
    int n = 0;
    SONAR_CLEAR_TRIGGERRX1;
    SONAR_CLEAR_TRIGGERRX2;
    SONAR_CLEAR_TRIGGERRX3;
    if (nSonar == 1)
        SONAR_SET_TRIGGERRX1; // Create low to high rising edge of
pulse
    else if (nSonar == 2)
        SONAR_SET_TRIGGERRX2; // Create low to high rising edge of
pulse
    else if (nSonar == 3)
        SONAR_SET_TRIGGERRX3; // Create low to high rising edge of
pulse

    _delay_us(30); //delay for at least 20us (but not more than 47ms)
to pass
    SONAR_CLEAR_TRIGGERRX1;
    SONAR_CLEAR_TRIGGERRX2;
    SONAR_CLEAR_TRIGGERRX3;

    int nEchoPin = -1;

    if (nSonar == 1)
        nEchoPin = SONAR_1_ECHOPW_PIN;
    else if (nSonar == 2)
        nEchoPin = SONAR_2_ECHOPW_PIN;
    else if (nSonar == 3)
        nEchoPin = SONAR_3_ECHOPW_PIN;

    if (nEchoPin == -1) // if due to some problem the echo pin is not
set return
        return 0;
    //int a = 4; //(1 << nEchoPin);

    while (!bit_get(SONAR_DATA_PIN, BIT(nEchoPin)));
    //while (!(SONAR_DATA_PIN & (1 << nEchoPin)))// >>
    (_BV(1<<nEchoPin))
    // int nQuit = 10000;
    // while (!(SONAR_DATA_PIN & _BV(SONAR_1_ECHOPW_PIN)) >>
    (SONAR_1_ECHOPW_PIN))
    //{
        // do nothing as long as EchoPW input is low
    //     if (nQuit--)
    //         return 0;
    //}

    n = 0;

    while (bit_get(SONAR_DATA_PIN, BIT(nEchoPin)))// >>
    (_BV(1<<nEchoPin))
    // while ((SONAR_DATA_PIN & _BV(SONAR_1_ECHOPW_PIN)) >>
    (SONAR_1_ECHOPW_PIN))
    {
        _delay_us(10); //delay of 10us
    }
}
```

```
//          DELAY(200); //delay of 10us
          n++;          // add 1 to n as long as EchoPW input is
high
    }
        //when we get here, the falling edge has occurred
    int dist = (10*n)/147;
    if (dist > 200)
        return 200; // the EchoPW pin remains high for upto 37.5ms
if no target is detected
    else
        return dist;
}
```

Compass.h

```
#ifndef __COMPASS_H__
#define __COMPASS_H__

void Compass_Init(void);
void Compass_GetAxes(void);
void Compass_ShiftOut(unsigned int nCmd);
unsigned int Compass_ShiftIn(int nBits);

#endif // __COMPASS_H__
```

Compass.c

```
#include <avr/io.h>
#include "Global.h"
#include "timer.h"
#include "Compass.h"

#define COMPASS_CLK_HI          bit_set(COMPASS_DATA_PORT,
BIT(COMPASS_CLK))
#define COMPASS_CLK_LO          bit_clear(COMPASS_DATA_PORT,
BIT(COMPASS_CLK))

#define COMPASS_EN_HI           bit_set(COMPASS_DATA_PORT,
BIT(COMPASS_EN))
#define COMPASS_EN_LO           bit_clear(COMPASS_DATA_PORT,
BIT(COMPASS_EN))

#define COMPASS_DIN_HI          bit_set(COMPASS_DATA_PORT,
BIT(COMPASS_DIN))
#define COMPASS_DIN_LO          bit_clear(COMPASS_DATA_PORT,
BIT(COMPASS_DIN))

void Compass_Init(void)
{
    bit_set(COMPASS_DATA_DDR, BIT(COMPASS_DIN));    // Set as output
pin
    bit_set(COMPASS_DATA_DDR, BIT(COMPASS_EN));      // Set as
output pin
    bit_set(COMPASS_DATA_DDR, BIT(COMPASS_CLK));    // Set as output
pin
    bit_clear(COMPASS_DATA_DDR, BIT(COMPASS_DOUT)); // Set as input
pin
}
void Compass_GetAxes(void)
{
    int Reset = 0x00; //command sent Reset 0000
    int Measure = 0x01; //command sent start meausrement 1000
    int Report = 0x03; //command sent report measurement 1100

    int Ready = 0x0C; //status returned 1100
    int xLow = 0;
    int xHigh = 0;
    int yLow = 0;
    int yHigh = 0;
    //clock in reset 0000
    COMPASS_EN_HI;
    DELAY(CLOCK_TICKS);
    COMPASS_EN_LO;
    Compass_ShiftOut(Reset);
    COMPASS_EN_HI;
    DELAY(CLOCK_TICKS);

    //clock in measure 1000
    COMPASS_EN_LO;
    Compass_ShiftOut(Measure);
    int Status;
    do
```

```

    {
        //clock in report 1100
        COMPASS_EN_LO;
        DELAY(CLOCK_TICKS);
        COMPASS_EN_HI;
        DELAY(CLOCK_TICKS);
        COMPASS_EN_LO;
        DELAY(CLOCK_TICKS);
        Compass_ShiftOut(Report);
        Status = Compass_ShiftIn(4);
//      lcdGotoXY(0, 0);
//      rprintfProgStrM("Status:");           rprintfNum(10,5,0,'
',Status);
        if (Status == 15)
        {
            COMPASS_EN_HI;
            DELAY(CLOCK_TICKS);
            COMPASS_EN_LO;
            Compass_ShiftOut(Reset);
            COMPASS_EN_HI;
            DELAY(CLOCK_TICKS);

            //clock in measure 1000
            COMPASS_EN_LO;
            Compass_ShiftOut(Measure);
            PORTB=~PORTB;
            timerPause(1000);
        }
    } while (Status != Ready);

    xLow = Compass_ShiftIn(8);
    xHigh = Compass_ShiftIn(3);
    yLow = Compass_ShiftIn(8);
    yHigh = Compass_ShiftIn(3);
    //lcdGotoXY(0, 0);
    //rprintfProgStrM("xLow:");           rprintfNum(10,5,0,' ',xLow);
    //rprintfProgStrM("  xHigh:");       rprintfNum(10,5,0,'
',xLow);
    //lcdGotoXY(0, 1);
    //rprintfProgStrM("yLow:");           rprintfNum(10,5,0,' ',xLow);
    //rprintfProgStrM("  yHigh:");       rprintfNum(10,5,0,'
',xLow);
}

void Compass_ShiftOut(unsigned int nCmd)
{
    int nBitPos = 0;
    for (nBitPos = 0; nBitPos < 4; nBitPos++)
    {
        (0x01&(nCmd>>nBitPos))? COMPASS_DIN_HI : COMPASS_DIN_LO;
        DELAY(CLOCK_TICKS);
        COMPASS_CLK_HI;
        DELAY(CLOCK_TICKS);
        COMPASS_CLK_LO;

//      while (nClock < CLOCK_TICKS)

```

```
//      {
//          if (nClock == 0)
//              bit_set(COMPASS_DATA_PORT, BIT(COMPASS_CLK));
//          nClock++;
//          if (nClock == CLOCK_TICKS/2)
//              bit_clear(COMPASS_DATA_PORT, BIT(COMPASS_CLK));
//      }
//      DELAY(CLOCK_TICKS);
//  }

unsigned int Compass_ShiftIn(int nBits)
{
    //  int nBitPos = 0;
    //  unsigned int nData = 0;

    while(nBits--)
    {
        //      int nClock = 0;
        //      bit_set(COMPASS_DATA_PORT, BIT(COMPASS_CLK));
        //      while (nClock < CLOCK_TICKS)
        //      {
        //          if (nClock == 0)
        //              bit_set(COMPASS_DATA_PORT, BIT(COMPASS_CLK));
        //          nClock++;
        //          if (nClock == CLOCK_TICKS/2)
        //              bit_clear(COMPASS_DATA_PORT, BIT(COMPASS_CLK));
        //      }
        //      COMPASS_CLK_HI;
        //      DELAY(CLOCK_TICKS);
        //      COMPASS_CLK_LO;
        //      nData = (nData << 1) | (bit_get(COMPASS_DATA_PIN,
COMPASS_DOUT) ? 0x01 : 0x00);
        //  }
        //  return nData;
    }
}
```

Makefile

```
#####  
#####  
# Makefile for the project Ant  
#####  
#####  
  
## General Flags  
PROJECT = Ant  
MCU = atmega128  
TARGET = Ant.elf  
CC = avr-gcc.exe  
  
## Options common to compile, link and assembly rules  
COMMON = -mmcu=$(MCU)  
  
## Compile options common for all C compilation units.  
CFLAGS = $(COMMON)  
CFLAGS += -Wall -g -c -Wstrict-prototypes -  
DF_CPU=14745600UL -Os -funsigned-char -funsigned-bitfields -fpack-  
struct -fshort-enums  
CFLAGS += -MD -MP -MT $(*)F.o -MF dep/$(@F).d  
  
## Assembly specific flags  
ASMFLAGS = $(COMMON)  
ASMFLAGS += $(CFLAGS)  
ASMFLAGS += -x assembler-with-cpp -Wa,-gdwarf2  
  
## Linker flags  
LDFLAGS = $(COMMON)  
LDFLAGS +=  
  
## Intel Hex file production flags  
HEX_FLASH_FLAGS = -R .eeprom  
  
HEX_EEPROM_FLAGS = -j .eeprom  
HEX_EEPROM_FLAGS += --set-section-flags=.eeprom="alloc,load"  
HEX_EEPROM_FLAGS += --change-section-lma .eeprom=0 --no-change-warnings  
  
## Include Directories  
INCLUDES = -I"E:\Homework\IMDL\Programming\Ant\" -  
I"E:\Homework\IMDL\Programming\Ant\..\..\avr\lib"  
  
## Libraries  
LIBS = -lm  
  
## Objects that must be built in order to link  
OBJECTS = Ant.o Sonar.o Motor.o Compass.o lcd.o rprintf.o timer.o  
nmea.o gps.o uart2.o buffer.o  
  
## Objects explicitly added by the user  
LINKONLYOBJECTS =  
  
## Build
```

```
all: $(TARGET) Ant.hex Ant.eep size

## Compile
Ant.o: ./Ant.c
    $(CC) $(INCLUDES) $(CFLAGS) -c $<

Sonar.o: ./Sonar.c
    $(CC) $(INCLUDES) $(CFLAGS) -c $<

Motor.o: ./Motor.c
    $(CC) $(INCLUDES) $(CFLAGS) -c $<

Compass.o: ./Compass.c
    $(CC) $(INCLUDES) $(CFLAGS) -c $<

lcd.o: ../../../../avrlib/lcd.c
    $(CC) $(INCLUDES) $(CFLAGS) -c $<

rprintf.o: ../../../../avrlib/rprintf.c
    $(CC) $(INCLUDES) $(CFLAGS) -c $<

timer.o: ../../../../avrlib/timer.c
    $(CC) $(INCLUDES) $(CFLAGS) -c $<

nmea.o: ../../../../avrlib/nmea.c
    $(CC) $(INCLUDES) $(CFLAGS) -c $<

gps.o: ../../../../avrlib/gps.c
    $(CC) $(INCLUDES) $(CFLAGS) -c $<

uart2.o: ../../../../avrlib/uart2.c
    $(CC) $(INCLUDES) $(CFLAGS) -c $<

buffer.o: ../../../../avrlib/buffer.c
    $(CC) $(INCLUDES) $(CFLAGS) -c $<

##Link
$(TARGET): $(OBJECTS)
    $(CC) $(LDFLAGS) $(OBJECTS) $(LINKONLYOBJECTS) $(LIBDIRS)
    $(LIBS) -o $(TARGET)

%.hex: $(TARGET)
    avr-objcopy -O ihex $(HEX_FLASH_FLAGS) $< $@

%.eep: $(TARGET)
    -avr-objcopy $(HEX_EEPROM_FLAGS) -O ihex $< $@ || exit 0

%.lss: $(TARGET)
    avr-objdump -h -S $< > $@

size: ${TARGET}
    @echo
    @avr-size -C --mcu=${MCU} ${TARGET}

## Clean target
.PHONY: clean
clean:
```

```
-rm -rf $(OBJECTS) Ant.elf dep/* Ant.hex Ant.eep  
## Other dependencies  
-include $(shell mkdir dep 2>/dev/null) $(wildcard dep/*)
```

The rest of the files namely lcd.c, lcd.h, rprintf.c, rprintf.h, timer.c, timer.h, buffer.c, buffer.h, gps.c, gps.h, nmea.c, nmea.h, uart2.c, uart2.h were used from avrlib so have not been included in the code.