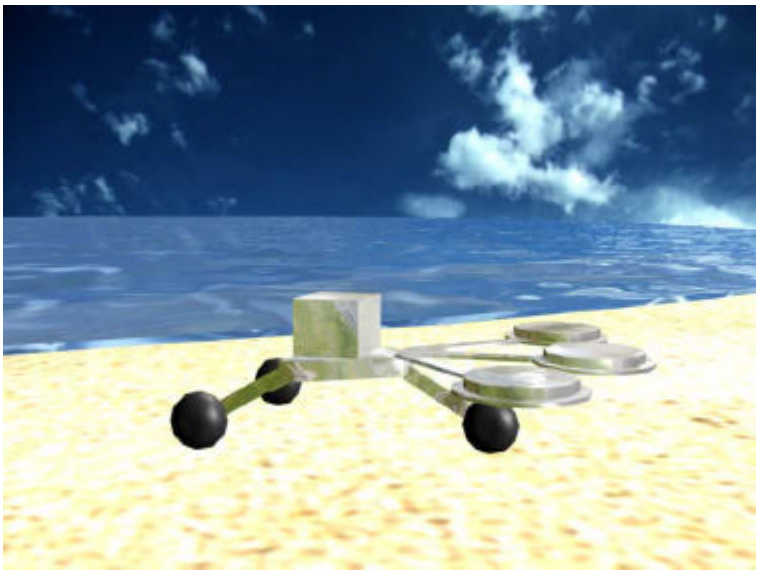
Atocha Too



Donald MacArthur

Center of Intelligent Machines and Robotics
&
Machine Intelligence Laboratory

Intelligent Machines Design Laboratory
EEL 5666C

TABLE OF CONTENTS

ABSTRACT

The “Atocha Too” is an autonomous robot that is used to find metallic objects on the east coast beaches of Florida. The robot operates using a Motorola microprocessor, servos, and various common robotic parts. The “Atocha Too” incorporates an Ultrasonic Ranging module and Infrared emitters and receivers for obstacle avoidance. The behaviors that were incorporated into the project allow for the “Atocha Too” to perform surveying and mapping of an area for obstacles and metal objects.

EXECUTIVE SUMMARY

The “Atocha Too” is an autonomous robot that was designed to search for buried treasure on the beaches of Florida’s East coast. Designed to be an aid to beach combers, the “Atocha Too” can also be used to survey and area for obsticals.

The robot uses many sensors to allow for the robot to better “feel” it’s environment. The robot incorporates an Ultrasonic Ranging board, and Infrared detectors and emitters to allow the robot to see it’s environment. The metal detecting circuit that is used on the robot give the robot the ability to sense metal objects.

The robot is controlled by a Motorolla HC11 microcontroller. This acts as the brains of the robot. The HC11 was programmed using Image Craft C. All behaviors and sensors were integrated into the HC11 using C programming code.

INTRODUCTION

The purpose of this report is to present the development and the construction of the “Atocha Too”. The robotic entity employs several types of sensors that allow the robot to traverse its surroundings and better perform its duties according to the environment. This report will discuss the different types of sensors, actuators, and programming that was used for the project.

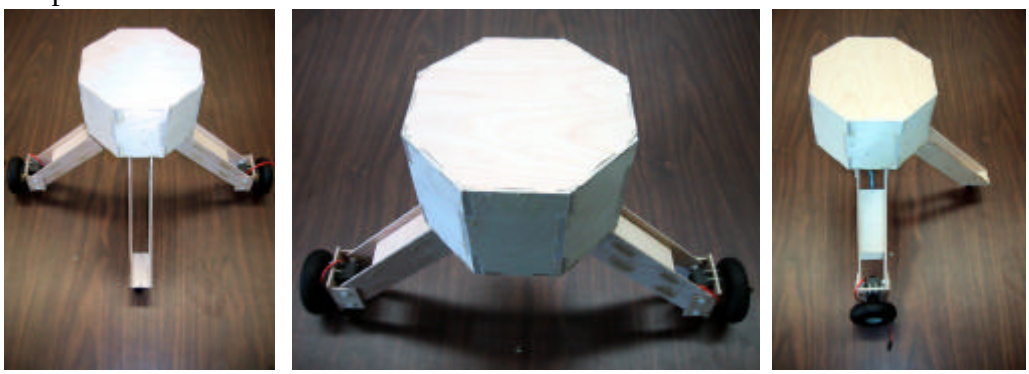
INTEGRATED SYSTEM

The organization of the robot was arranged so that given the constraints of the microcontroller, the different functions of the robot could be performed without interaction or confusion. The robot operated by first sensing the environment, then performing actions accordingly to the perceived data.

The microcontroller controlled the actuation, sensors, and decision processes of the robot. The robot would sense the environment, decide on a course of action, and then actuate the robot accordingly. The microcontroller integrated sensory data from the Ultrasonic Ranging board, the IR receivers, and from the Metal detecting circuits. All of the sensory data was recorded so that the information could be used latter for mapping or surveying.

MOBILE PLATFORM

The platform that was used for the robot is illustrated below.

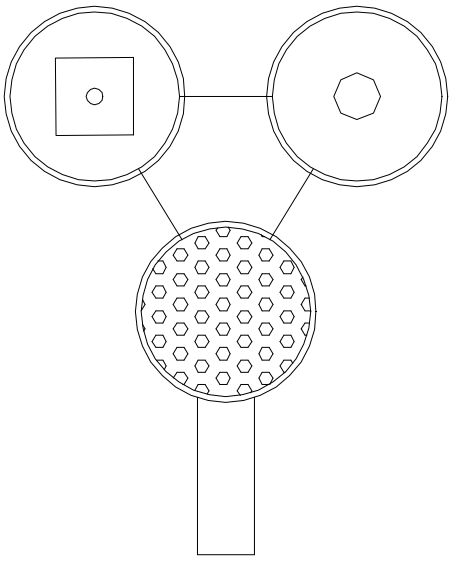


The mobile platform utilizes two modified servos for main drive power, with a rear caster. Changing the direction or speed, of the left and right wheels, controled the robot's direction. The left and right legs of the robot, incorporates a suspension system which was used to overcome the pitfalls of the tricycle design.

ACTUATION

Generating a PWM signal from the microcontroller actuated the main drive wheels. The rotation of the wheels was controlled by modifying the PWM signal, which were associated with reverse and forward motion commands.

Another servo was used to control the panning head of the robot. The panning head of the robot contains the Sonar transducer, and the IR emitter and receiver. An illustration of the panning head is shown below.



The servo was connected to a gear train with a 2:1 ratio which was then connected to the panning head. This allowed for the 180 degree rotation of the servo to cause a 360 degree rotation of the panning head. Actuating this servo allowed for sensor data to be taken around the periphery of the robot.

SENSORS

There were several types of sensors that were used on the Atocha Too. The organization of the sensors is shown below in Figure 1.

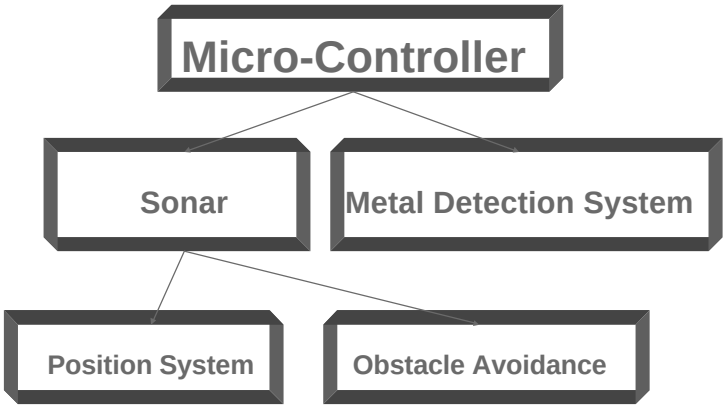


Fig.1

The micro-controller acts as the center were all sensor data is collected and processed. A TJ pro main board was used as the micro-controller to control and process all sensors and actions of the robot. The current sensors consist of a metal ordinance detector and a sonar ranging system which are shown below as Figure 2.

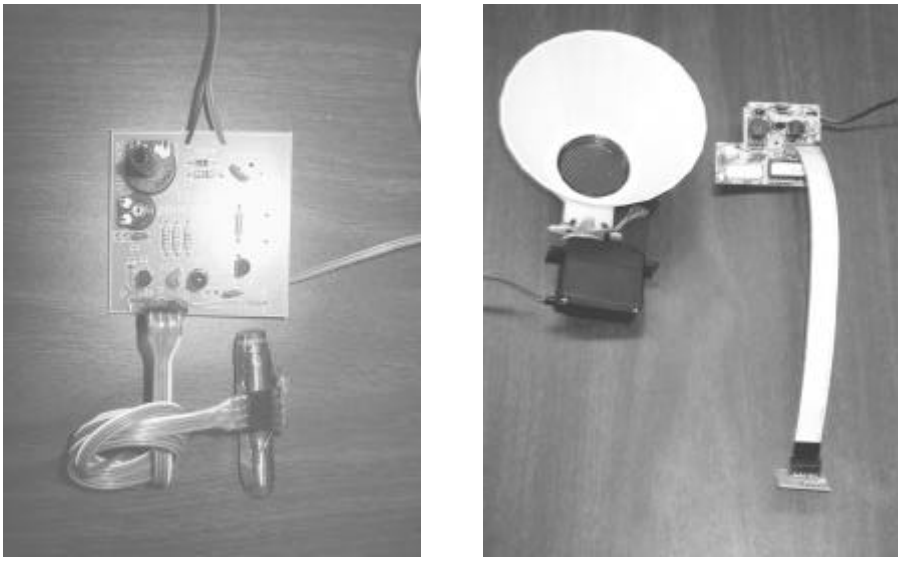


Fig. 2

SONAR

The Atocha Too will be primarily used in an outdoor environment where the use of IR detectors will be useless if they are flooded with sunlight. Therefore, sonar was used as the primary sensor for obstacle avoidance and obstacle ranging. The sonar modules that were used are Polaroid 6500 Ultrasonic Ranging Boards with the corresponding transducers. These modules have an ultimate range of ~35 feet. The sonar modules will be used for obstacle avoidance and an Ultrasonic Positioning System.

The SPS (Sonar Positioning System) is currently under development. All hardware required for the system is available, and all that is required for a complete system is software integration and further testing.

The sonar system operates on the premise that sound travels at a constant rate in air. The system produces an Ultrasonic chirp that bounces off of any object lying in front of the transducer. By timing the duration between the initiation of the chirp and the received signal, the distance from the transducer and the object can be obtained. Figure 3 shown below demonstrates ranging data taken using the sonar modules.

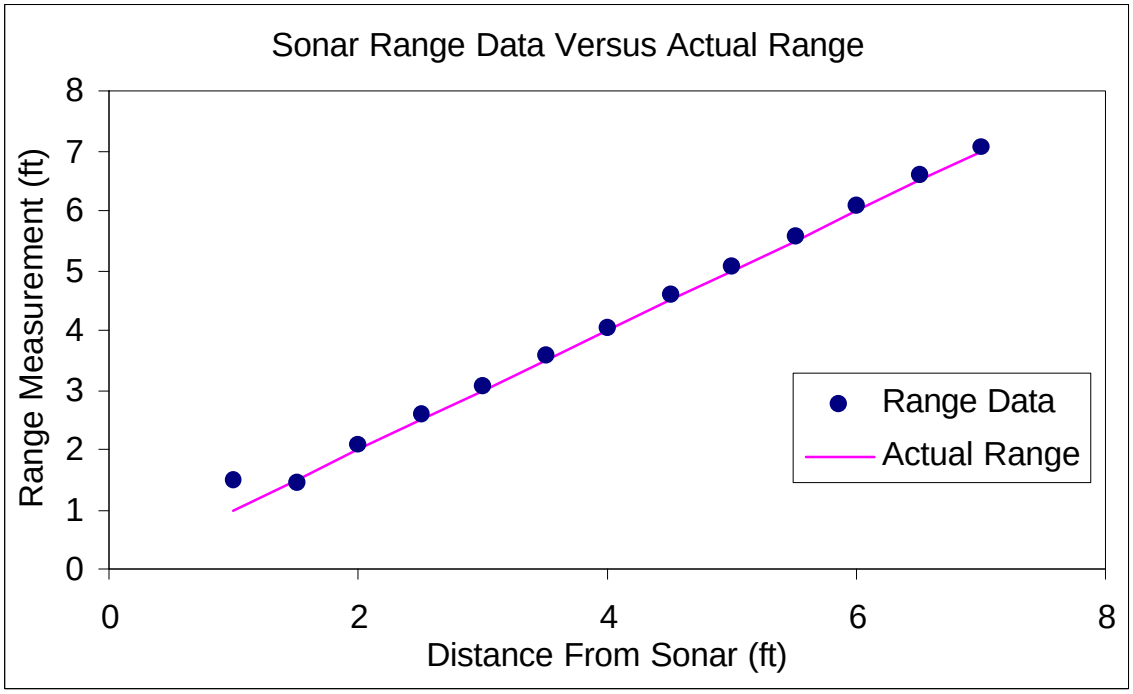


Fig. 3

There exists an inherent flaw in the sonar modules. The minimum range that the transducers can operate is ~1.6 feet. This is caused by the fact that the transducers still vibrate immediately after the Ultrasonic chirp is initiated. This can cause noise on the receive line which can be interpreted as a receive signal at close distances.

Therefore the sonar modules are suited for operations where the robot will not be expected to require low range data. Figure 4 shown below shows the configuration of the sonar modules integrated with a Tjpro board.

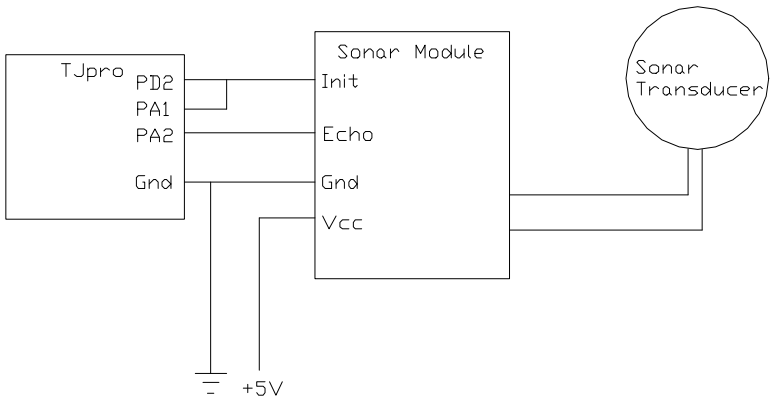


Fig. 4

The sonar module operates by first receiving an Init signal. The sonar transducer is then sent a signal, and the Echo line goes high when a signal is received. The sonar system utilizes the input capture routines of the 68HC11 micro-controller.

By multiplying the difference between the times that PA1 and PA2 go high by the propagation speed of sound through air, the range of an object can be obtained. This forms the basis of the sonar ranging system.

METAL DETECTION

The metal detection circuit was constructed from a kit. The circuit is designed to detect metal objects in walls and homes to avoid electrocution while performing home modifications. The circuit has an LED that lights when a metal object is placed close to the search coil.

By placing taps across the LED, an analog signal can be drawn from the circuit that relates to metal detection. The voltage across the LED varies from .6-.8 volts when no object is close to the search coil, and 2-2.4 volts when a metal object is against the search coil. These taps are connected to an analog port on the TJ-pro board. This allows for the TJ-pro board to receive and process metal detection sensor data and act accordingly.

BEHAVIORS

The robot's behaviors consist of auto-calibration, obstacle avoidance, metal search, and surveying behaviors. When the robot is first turned on, the auto-calibration behavior sets threshold values for the IR detector, and metal detector circuit so that the robot can perform it's functions in environments with varying IR and metal conditions.

The obstacle avoidance behavior utilizes sensor data from multiple positions of the panning sensor head and maneuvers the robot to avoid obstacles.

The metal search behavior reads the input from the metal detecting circuit and spins the robot to indicate to the user that the robot has found a metal item.

The surveying behavior actuates the panning sensor head and records the corresponding sensor data into vectors that can be used for mapping and surveying purposes.

EXPERIMENTAL LAYOUT

The Sonar experiments were conducted with the vehicle completely assembled. Obstacles were placed a measured distance from the robot and the range data returned from the robot was recorded. Without any modifications made to the constants or hardware, the accuracy for the Sonar was on the order of a fraction of an inch.

The metal detector experiments were conducted with the metal detecting circuit separated from the robot. The analog taps were used to measure the voltage as various metallic object were placed near the inductive coil of the metal detecting circuit.

CONCLUSION

This paper has discussed the development and systems of the “Atocha Too”. This project has inspired several other projects for Graduate research. Among those is the SPS system, which still needs to be constructed and implemented into a working system. The IMDL class has inspired my interest with robotics and electronics and has cause me to shift my research interests into this field.

APPENDIX

ICC code for the “Atocha Too”

```
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

        Title       FINAL3.c
        Programmer     Donald MacArthur

        Date        12/20/2000
        Version        3

        Description
        This program incorporates all of the functionality of the Atocha Too
        and organized the software for modularity and clarity
        Systems:
                -BRAINS
                        -Behaviors
                        -Sensors
                                -IR
                                -Sonar
                                -Metal Detector
                        -Motor Control
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/

/************************** Includes *******************************/

#include <tjpbase.h>
#include <stdio.h>
#include <mil.h>
#include <hc11.h>
#include <math.h>


/************************ End of Includes ***************************/


/************************** Constants *******************************/

#define SONAR_AVOID_THRESHOLD        2500
#define IR_head                             analog(2)             //IR sensor located on
panning head
#define IR_BIAS                                     10                    //IR
additional value of obsticle
#define LOOP_COUNT                          5000          //# of loops before
LOOK_FOR_METAL exits
#define METAL_THRESH                5                     //threshold for metal
detector



/************************ End of Constants **************************/

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//Global Variables
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

        int speedr, speedl, angle,CW,ok;

        int PWpan;                  //Panning head servo global Pulse Width

        int sonar_range_array[8], IR_range_array[8];

        int     IR_cal_const;               //calibration constant for IR sensor

        int metal_cal_const;        //calibration constant for metal detector

        int COUNT;                                  //loop counter

        int found_metal;                    //flag if metal is found
```

```
        float distance, thresh;

        unsigned int   command_message, status_message;            //message sets for
robot
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~



/*************************** Prototypes *********************************/
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//Initialization Functions
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
void    INITIALIZE();                //initialize functions and global variables

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//BRAIN Functions
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
void    SENSE();

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//Motor Control Functions
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

void    turn();
void    SPIN();
void    STOP();
void    GO();

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//Sensor Functions
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


//Sonar Functions
void    init_sonar();                //Initialize Sonar Registers
void    sonar_ping();                //Start Sonar Ping and allow input capture to occur
int     sonar_range();               //Calculate Range of Obsticle



//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//Behavior Functions
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

void SPIN(void);
void LOOK_FOR_METAL();


/************************ End of Prototypes *****************************/

void main(void)
/****************************** Main ***********************************/
{


        INITIALIZE();
        START;  /*Press the rear bumper to start the program*/

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        BRAIN LOOP
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/

   while(1)
  {
        ok=0;

        while(!ok)
        {
                SENSE();
```

```
                if((sonar_range_array[2]<=SONAR_AVOID_THRESHOLD)||
                        (sonar_range_array[3]<=SONAR_AVOID_THRESHOLD)||
                        (sonar_range_array[4]<=SONAR_AVOID_THRESHOLD))

                {
                        turn();
                        STOP();
                }
                else ok=1;
        }

        ok=0;
        COUNT=0;
        found_metal=0;
        while(!ok)
        {
                LOOK_FOR_METAL();
                COUNT++;
                if((COUNT>LOOP_COUNT)||(found_metal==1))
                {
                        ok=1;
                        GO();
                        wait(500);
                        STOP();
                }
        }
        STOP();

  }
}
/***************************** End of Main ******************************/

void turn()
/*********************************************************************
 * Function:   This function will avoid an
 *                           obstacle when found by the Sonar, and turn away
 * Returns:  None                                                      *
 *                                                                     *
 * Inputs                                                              *
 *   Parameters:                                                    *
 *   Globals:    None                                                  *
 *   Registers:  TCNT                                                  *
 * Outputs                                                             *
 *   Parameters: None                                                  *
 *   Globals:    None                                                  *
 *   Registers:  None                                                  *
 * Functions called: motorp(), wait()                               *
 * Notes:                                                              *
 *********************************************************************/
{
 int i;
 unsigned rand;

 rand = TCNT;


   motorp(RIGHT_MOTOR, -MAX_SPEED);
   motorp(LEFT_MOTOR, MAX_SPEED);

 i=(rand % 1024)*5;
 if(i>250) wait(i+250); else wait(250);


}

/***********************End Function turn ***************************/

void init_sonar()
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        Function:                    None
```

```
        Description:          None

        Returns:                      None

        Inputs:                   None
        Parameters:         None
        Globals:              None
        Registers:          None
        Outputs             None
        Parameters:           None
        Globals:              None
        Registers:            None
        Functions called:     None

        Notes:              None
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/

{
        SET_BIT(DDRD,0x4);
        SET_BIT(TCTL2,0x14);
        CLEAR_BIT(TCTL2,0x28);
}

void INITIALIZE()
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        Function:                     INITIALIZE

        Description:          initializes Global variables and TJPro init functions

        Returns:                      None

        Inputs:                   None
        Parameters:         None
        Globals:              None
        Registers:          None
        Outputs             None
        Parameters:           None
        Globals:              None
        Registers:            None
        Functions called:

                        init_analog()
                        init_motortjp()
                        init_clocktjp()
                        init_servotjp()
                        init_sonar()

        Notes:              None
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
{
        init_analog();
        init_motortjp();
        init_clocktjp();
        init_servotjp();
        init_sonar();
        PWpan=1000;
        ok=1;

        IRE_ON;
        IR_cal_const=IR_head;

        //calibrate metal detector
        metal_cal_const=analog(3);

}



void sonar_ping()
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
        Function:                       None

        Description:            None

        Returns:                        None

        Inputs:                     None
        Parameters:         None
        Globals:                None
        Registers:          None
        Outputs             None
        Parameters:             None
        Globals:                None
        Registers:              None
        Functions called:       None

        Notes:              None
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/

{
        TFLG1=0x0;
        SET_BIT(PORTD,0x4);
        wait(30);
        CLEAR_BIT(PORTD,0x4);
}

int sonar_range()
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        Function:                       sonar_range

        Description:            checks the flag TFLG1 to see if a signal was received

        Returns:                        float which represents the distance to obstical

        Inputs:                     None
        Parameters:         None
        Globals:                None
        Registers:          None
        Outputs             None
        Parameters:             None
        Globals:                None
        Registers:              None
        Functions called:       None

        Notes:              None
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/

{
        if (((TFLG1)&(0x4))==0)
        return -1;
        else
        if((((TIC2-TIC1)>>1)*.569)<0) return -(((TIC2-TIC1)>>1)*.569);
        else return(((TIC2-TIC1)>>1)*.569);
}


void SENSE()
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        Function:                       SENSE

        Description:            This functions will pan the head servo and save the
                                        sensor values into arrays for examination.

        Returns:                        None

        Inputs:                     None
        Parameters:         None
        Globals:                sonar_range_array, IR_range_array, PWpan
        Registers:          None
        Outputs             None
        Parameters:             None
```

```
        Globals:              PWpan
        Registers:            None
        Functions called:     sonar_range()

        Notes:              None
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
{
        //local variables
        int     i,j;            //Generic Counter
        int CCW;        //Rotation


        if(PWpan==1000)         CCW=1;

                else                    CCW=0;

                //Pan head servo
                if(CCW==1)              //if counter clockwise rotation is preferred
                {
                        for(i=1; i<=8; i++)
                        {
                                sonar_ping();
                                sonar_range_array[i-1]=sonar_range();
                                IR_range_array[i-1]=IR_head;

                                PWpan=1000+i*4000/8;
                                servo(0,PWpan);
                                wait(200);
                        }
                }
        else
        {
                        for(i=1; i<=8; i++)
                        {
                                sonar_ping();
                                sonar_range_array[8-i]=sonar_range();
                                IR_range_array[8-i]=IR_head;

                                PWpan=5000-i*4000/8;
                                servo(0,PWpan);
                                wait(200);
                        }

        }


        //print to com for testing
        printf("\nIR data:  ");
        for(i=1; i<=8; i++)
        {
                printf("  %d", IR_range_array[i-1]);
        }
        printf("\nSonar data:  ");
        for(i=1; i<=8; i++)
        {
                printf("  %d", sonar_range_array[i-1]);
        }
}



void LOOK_FOR_METAL()
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        Function:                       None

        Description:            None

        Returns:                        None

        Inputs:                     None
        Parameters:         None
```

```
        Globals:                None
        Registers:          None
        Outputs             None
        Parameters:             None
        Globals:                None
        Registers:              None
        Functions called:       None

        Notes:              None
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
{
        motorp(RIGHT_MOTOR, -MAX_SPEED);
        motorp(LEFT_MOTOR, -MAX_SPEED);
        if(analog(3)>metal_cal_const+METAL_THRESH)
        {
        found_metal=1;
        SPIN();
        }


}

void SPIN()
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        Function:                       None

        Description:            None

        Returns:                        None

        Inputs:                     None
        Parameters:         None
        Globals:                None
        Registers:          None
        Outputs             None
        Parameters:             None
        Globals:                None
        Registers:              None
        Functions called:       None

        Notes:              None
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
{
        motorp(RIGHT_MOTOR, -MAX_SPEED);
        motorp(LEFT_MOTOR, MAX_SPEED);
        wait(7000);


}
void STOP(void)
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        Function:                       None

        Description:            None

        Returns:                        None

        Inputs:                     None
        Parameters:         None
        Globals:                None
        Registers:          None
        Outputs             None
        Parameters:             None
        Globals:                None
        Registers:              None
        Functions called:       None

        Notes:              None
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/

{
```

```
        motorp(LEFT_MOTOR, 0);
        motorp(RIGHT_MOTOR, 0);

}

void GO(void)
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        Function:                     None

        Description:          None

        Returns:                      None

        Inputs:                    None
        Parameters:         None
        Globals:               None
        Registers:          None
        Outputs             None
        Parameters:            None
        Globals:               None
        Registers:             None
        Functions called:      None

        Notes:              None
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
{
        motorp(LEFT_MOTOR, -MAX_SPEED);
        motorp(RIGHT_MOTOR, -MAX_SPEED);
}
```