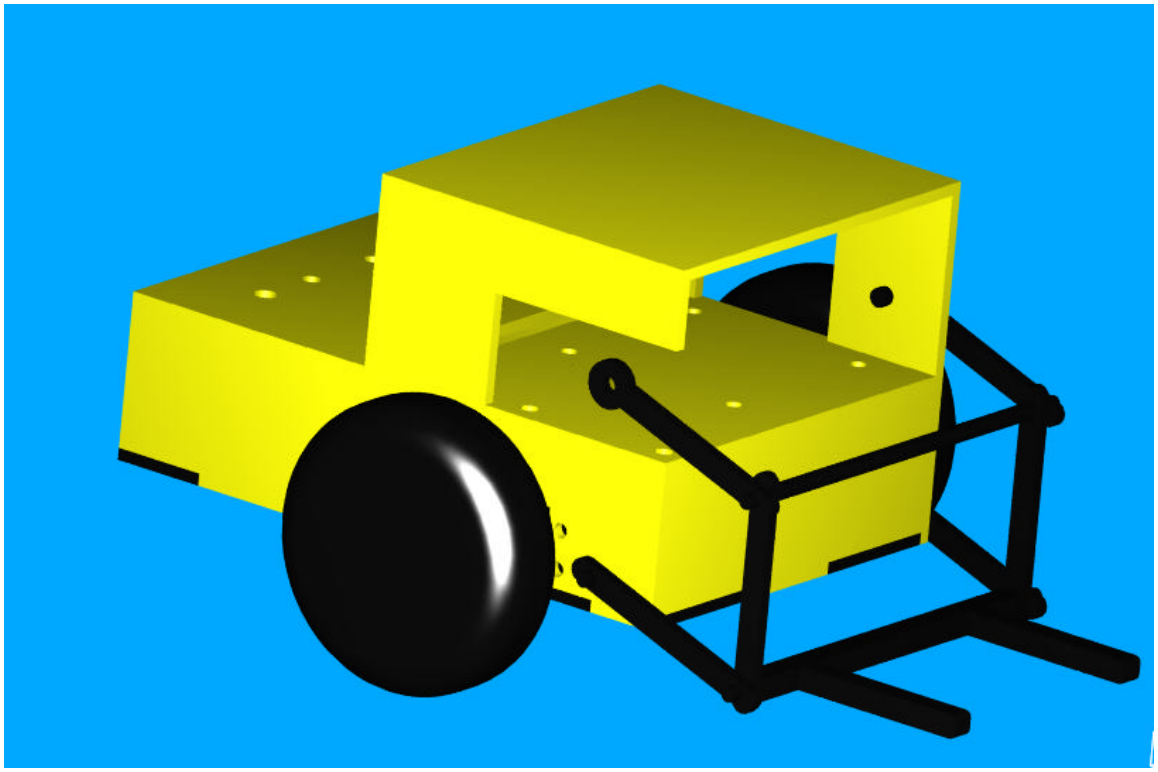
FORKER

Autonomous Miniature Forklift



Final Report

Presented to

Dr. Antonio Arroyo

by

Chad Tobler
23 April 2002

As partial fulfillment
for the requirements of
EEL5666C

Table of Contents

Abstract

This report details the construction and design of an autonomous miniature forklift. Detailed explanations are given for all of the hardware and software aspects of the vehicle. All sections of the integrated system are included: mobile platform design, actuation, sensors, and programmed behaviors are all explained. An analysis of the experimental layout and performance is also included. An appendix is attached containing the program code and other supplementary material.

Executive Summary

FORKER is an autonomous mini-forklift. FORKER will follow a black line in a model factory. When FORKER locates a loading location, he will execute a loading function to retrieve a load. He then goes back to the line and searches for the unloading station, where a similar unloading routine is executed. If FORKER detects and obstacle in his path, he will stop and beep until the obstacle is removed.

Servo motors are used for the actuation of the drive and fork systems, all controlled by the TJPRo 68HC11 board sold by the Mechatronix company. A variety of sensors are used as input to the robot, including photo reflectance sensors, 40 kHz IR detectors, and multiple switches. LEDs and a buzzer are used to give visual and audible feedback of the robots behavior.

The robot functions as designed, and is an impressive display of the potential to create a functional autonomous robot that performs a useful task.

Introduction

Autonomous mobile vehicles are becoming increasingly common in the world around us. Autonomous vehicles have been designed and implemented to perform a wide variety of tasks, from delivering medical sample in a hospital to sweeping and clearing unexploded ordnance from a mine-field. One of the most practical and popular applications for autonomous vehicles has been in the area of material handling. The use of autonomous material handling vehicles is now common in the high volume production facilities of many industries.

I chose to build an autonomous miniature forklift in order to increase my understanding of the technical issues and details involved in producing an autonomous material handling vehicle. The basic objectives of the project were to build a mini-forklift that could follow a painted line on a mini-factory floor, and load and unload cargo at designated locations. The forklift would also be capable of obstacle avoidance while maneuvering the course in the mini-factory. This paper will explore all aspects of the design and construction of this functional autonomous vehicle.

Integrated System

The forklift is designed to function in the mini-factory environment that will be set up for it. The system uses the Mechatronix TJPro board with the Motorola 68HC11 microcontroller for its brain. Various sensors provide input to the board in order to determine which behaviors to perform by actuating outputs devices such as LEDs, servos, and a beeper.

The model factory consists of a black electrical tape track the can be placed on nearly any smooth background surface. Initially, cardboard was going to be used for the background, but the line following sensors proved to be sensitive enough to distinguish the tape line from nearly any color background surface (except black). For this reason, the track can be laid using the tile floor of the room as its background surface. The track has one pick-up location, and one drop-off location. The loading locations are indicated by a tape cross on the track. The forklift maneuvers the course using line following to the next loading location. The line following program works well enough that the tape line can include curves and right angles.

Obstacle detection functions will prevent the forklift from colliding with other vehicles or obstructions while in the path following mode. Once the vehicle detects a loading point indicator, the obstacle avoidance routine will be disabled, and the forklift will advance and execute a loading routine. Once loaded, the forklift will return to the track and follow it until it detects the unload point indicator. After detecting the unload point, and unload routine will be performed, and the forklift then returns to the track to search for the next load. This scope and function of the vehicle could be easily expanded to include sorting and delivering to multiple locations based on load identification if time

allows.

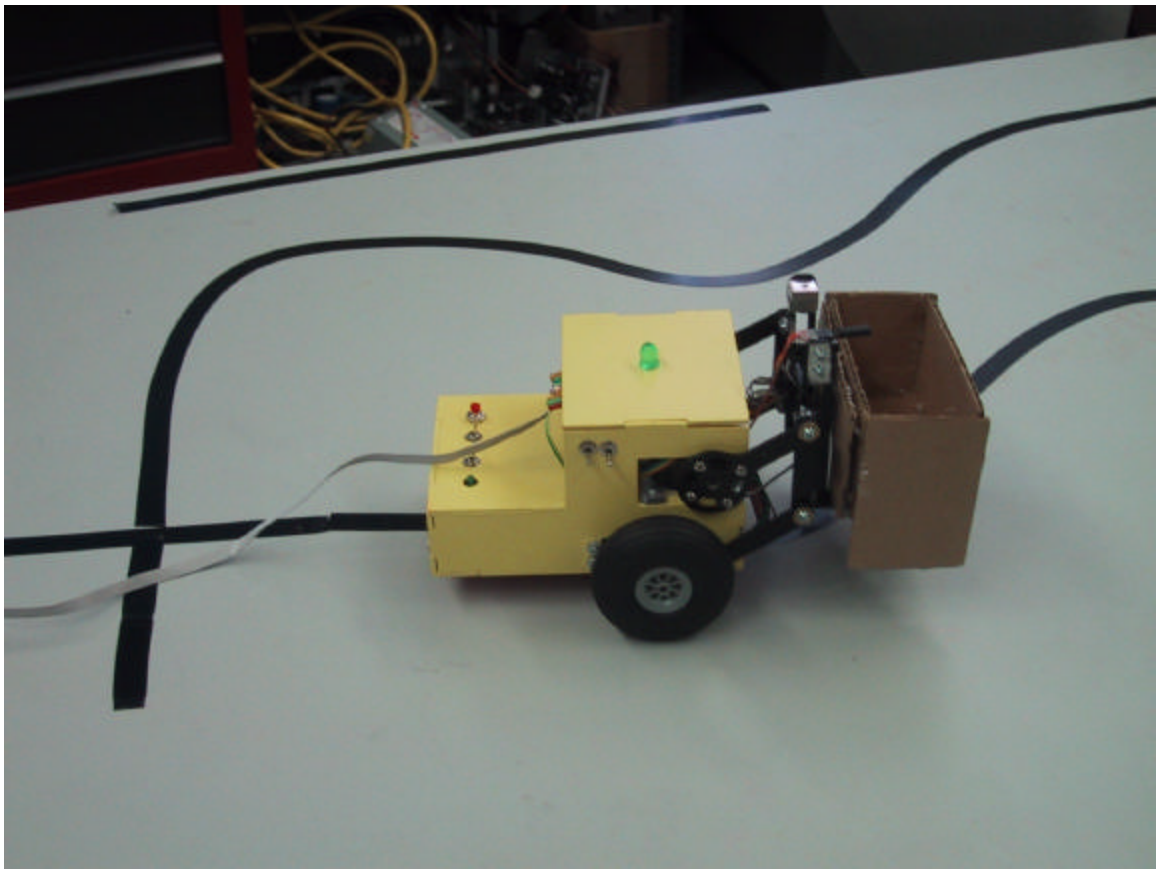


Figure 1: A picture of the loaded forklift following a line on a test course.

Mobile Platform

The mobile platform was designed to resemble an actual forklift. Two independently driven front wheels provide driving and steering capabilities, and a peg supports the rear end of the vehicle to maintain the platform level. The bulk of the platform is constructed of 5-ply model aircraft plywood. The platform was designed as small as possible to provide maximum maneuverability within the mini-factory. It was also designed to provide easy access to the processor board, sensors, driving and forklift

systems. The platform was also designed to allow the majority of the wires and electronics to be neatly concealed, resulting in a clean, aesthetically pleasing appearance.

Actuation

Actuation of the drive system is accomplished using hacked servo motors to drive each of the front wheels. Hacked servos are ideal for this application because they are geared to provide high torque at low speeds. Parallax continuous rotation 43 oz-in servos were ordered online from Parallax for this application. Because the forklift follows a track, low speeds will be acceptable, and the torque is needed to move the small loads. Also, using separate servos for each wheel will allow the forklift to rotate about an axis directly between the front wheels by rotating them in opposite directions. This feature is helpful in performing tight 90-degree turns.

Countless hours were spent trying to characterize the response of the hacked servo motors. I encountered countless problems while trying to get the motors to respond similarly using the motor control code provided by Mechatronix. In order for the forklift to drive straight, a command of 83 was sent to one motor and a command value of 1 was sent to the other (range of potential command values for forward direction was 0-100)! This proved to be a very frustrating process because I seemed to have so little ability to control the speed of the motors, due to their dissimilar, sporadic responses. After discussing the situation with the a teaching assistant, he suggested that I try using the servo control code provided by Mechatronix, which proved to do a much better job controlling the hacked servos.

The forklift mechanism will be actuated by a servomotor connected to an actuating link of a parallel four-bar lift mechanism. As the servomotor rotates the link

about the pivot point, the forks translate from ground level to approximately 3 in. above the ground, always maintaining the forks (and the load) parallel to the ground. The servo used was a standard Futuoba 43 oz-in servo obtained from ServoCity. The servo functions very well for this application, but the actuating a single side of the two-sided mechanism to raise the forks provided some unbalance of force in the mechanism. Future designs would actuate the mechanism from its center, instead of from one side as this initial

design.

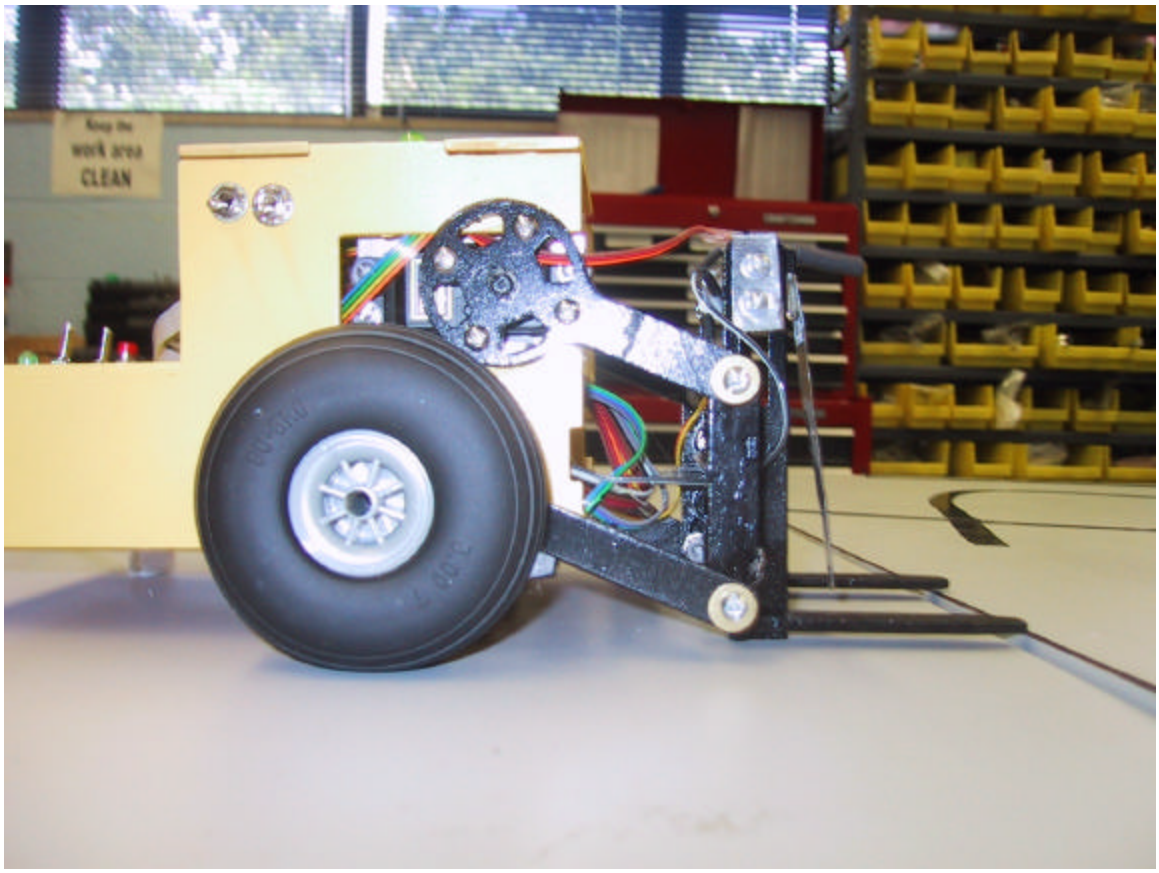


Figure 2: A picture of the four bar mechanism use to lift the forks. The servo is attached behind the round part of the top link.

An array of LED's and a buzzer are used to provide visual and audio feedback for the behavior being executed by the robot. The buzzer sounds when the forklift drives in reverse, to simulate the sounds of an actual forklift. An large LED mounted on the cab roof lights to indicate that the forklift is loaded, and flashes when the forklift is loaded and moving to the unloading station. Other auxiliary LEDs indicate if the forklift is driving straight, correcting left or right, turning left or right, or stopping.

Sensors

All eight of the analog input ports available on the TJPro board are used for interfacing a varied suite of sensors to the microcontroller. An Sharp 40 kHz modulated infrared detector and emitting LED is mounted on the top front of the fork assembly to detect and prevent collisions with close range obstructions. Silicon photo transistors are arranged in an array of three sensors pointed toward the ground to perform the line following task. These photo transistors function well to distinguish the black line from the lighter colored background. Two additional photo transistors have been added to be on the outside edges of the platform to detect the black crosses in the track, indicating a load or unload point. One more of the photo transistors is used on the forks to detect when the load is in place. A limit switch also is mounted on the forks as a redundant load sensor.

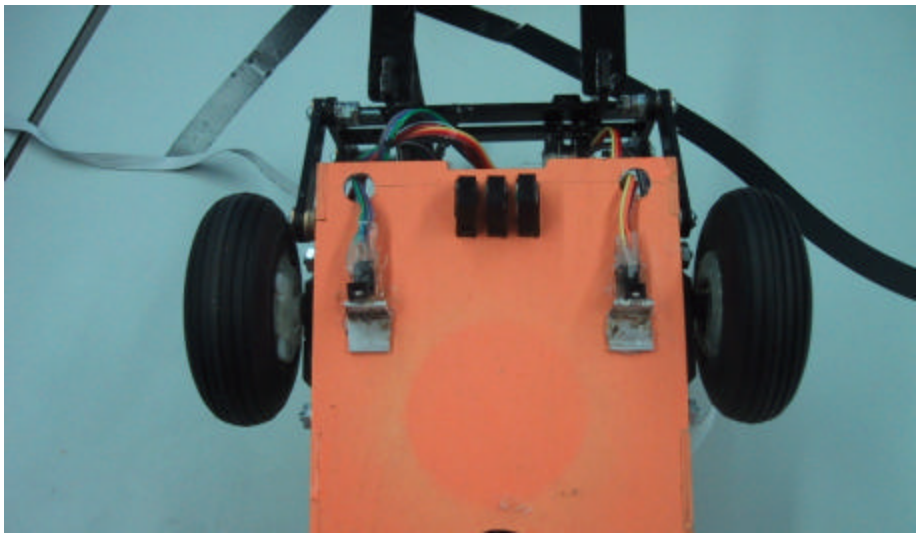


Figure 3: A photo of the underside of the robot showing the three line following and two edge photo reflectance sensors.

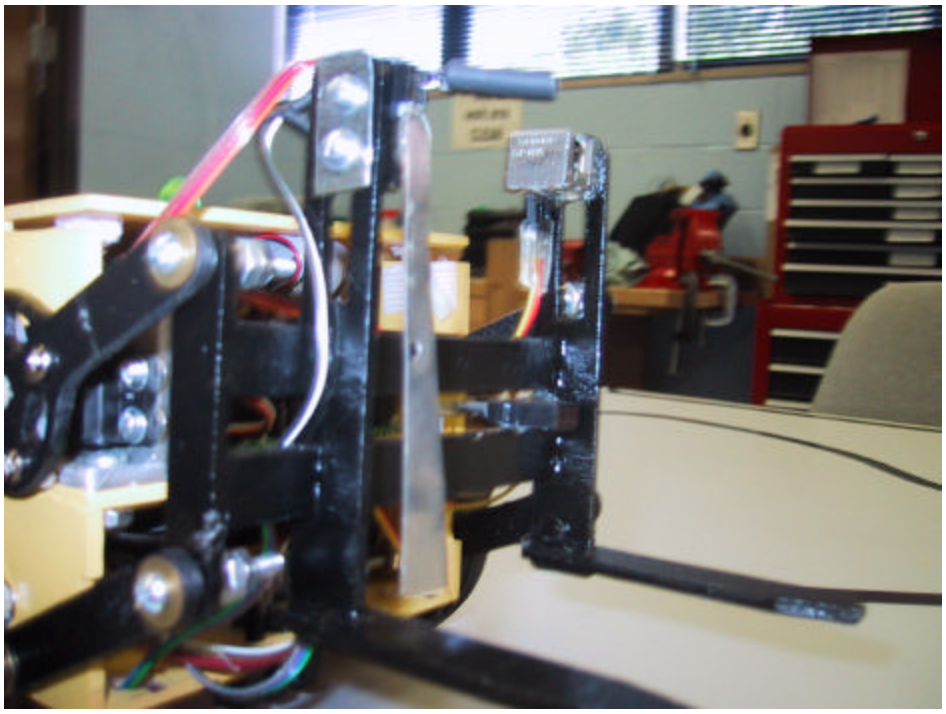


Figure 4: A photo showing the Sharp IR detector can, the load sensing limit switch, and the load sensing photo reflectance sensor.

In order to accomplish the task of following a line in a model factory, appropriate sensors were chosen and evaluated. Various types of sensors available for finding a

painted line on a factory floor include photocell light sensors and infrared reflectance sensors. The photocell light sensor is a resistor that changes resistance when exposed to varying amounts of light. The infrared reflectance sensor is a phototransistor that detects the wavelengths emitted by an infrared LED it is coupled with in a small package. I performed a large variety of experiments with both types of sensors, and the results and conclusions of that work are described in the following sectoins.

Sensor Background

Initially, I was planning on using Cadmium Sulfide photocells (CdS) as the sensors used to accomplish the task of line following. Class lectures and the lab teaching assistants recommended the CdS cells because they had been used successfully in the past for line following robots, and were readily available and inexpensive. After building the voltage divider circuits used with the CdS cell, I was disappointed by the huge amount of variance between individual units. I had differences in resistance of greater than 10 orders of magnitude between some of the identically constructed CdS sensors. Those differences were reduced by using different resistors for the voltage dividers, but I was still unsatisfied with the performance to the CdS sensors.

After some additional research into available photo detectors, I decided to try some models of infrared reflectance sensors. These used a matched pair of an infrared emitting LED and detecting phototransistor. They come packaged together in a small, neat package. I found the photo reflectance sensors to perform far superior than the traditional CdS cells. The results are presented in the following sections.

Experimental Apparatus and Hardware Configuration

The experiments were performed using the TJPro board, in similar conditions to the operating environment the sensors would be used in. The apparatus for testing the sensors was set up on a table in a lab with fluorescent lights and blind covered windows. In order to control the distance the sensor was above the ground, it was attached to a straight edge with indexed increments, and placed in a vice. The sensor was aimed at the tabletop, and raised from the surface in 0.10 inch increments. At each increment, brown, white, and black colored paper was placed under the sensor, and the value was recorded. This process was repeated three times for each sensor, in order to ensure the precision of the data. The distance the sensor was test over ranged from 0 to 1.5 inches above the table surface.

A 330 Ohm resistor was wired in series with the emitter side of the sensor to regulate the current through the LED. A 47 Kohm voltage divider was constructed for used with the detector side of the sensor. A 2.2 Kohm voltage divider was also tried, but produce a very reduce range of output values, so it was not used for further testing.

The CdS cell used was obtained from the IMDL stock room and was constructed using a 47 Kohm voltage divider. The photo reflectance sensors were order from Mouser Electronics. Two different models were test, namely the QRB1114 and the QRD1114. These sensors differed in the design of their package, and in the slightly in the current handling capacity of the transistor. They are available from Mouser for $1.30 and $1.13 respectively. Detailed information about them can be viewed in the appendix.

Experimental Design for Sensors

Tests were performed in order to compare and evaluate my best performing CdS cell, and the two models of photo reflectance sensors. The each unit of the different models was similarly tested over the 1.5 inch range (at 0.1 inch increments) over white, black, and brown surfaces. These experiments allowed the outputs of the different models of sensors to be compared over the variable colors and distances. The testing of multiple units also allowed the different units of the same model to be compared.

Sensor Testing Results

The results of the sensor experiments are plotted and compared below, because the graphical representation of the data is much easier to interpret.

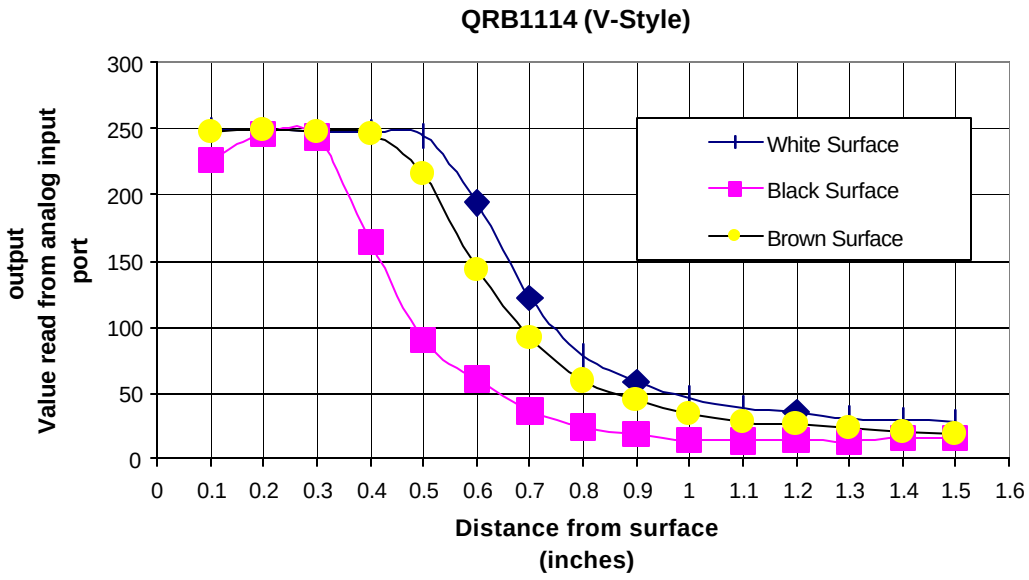


Figure 5: This graph compares the output of the QRB1114 sensor over white, black and brown surfaces.

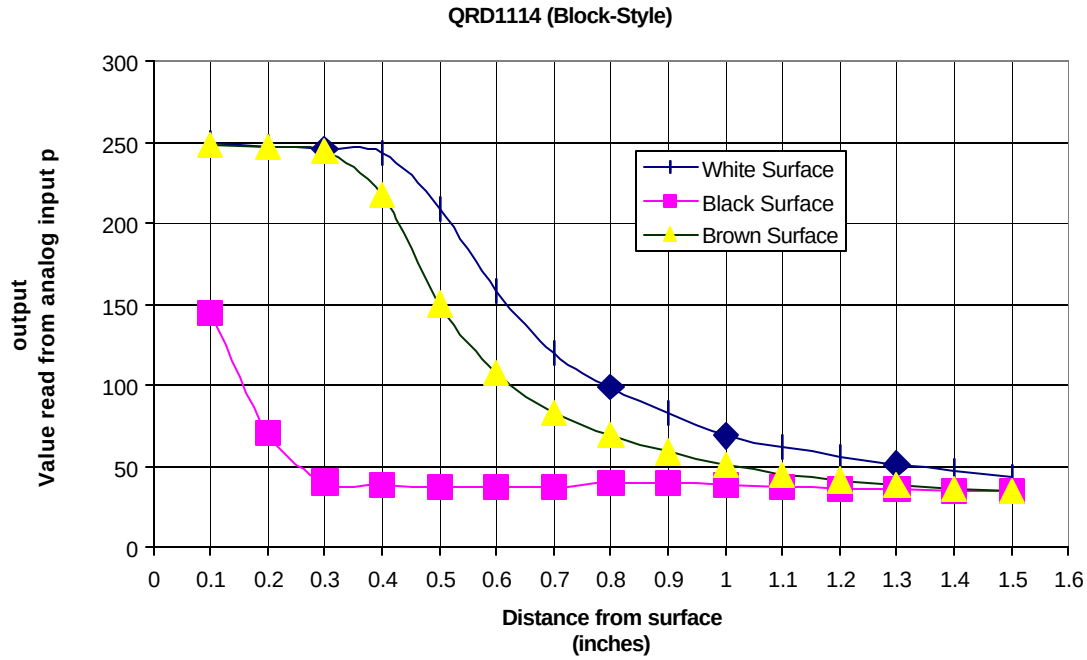


Figure 6: This graph compares the output of the QRD1114 sensor over white, black and brown surfaces.

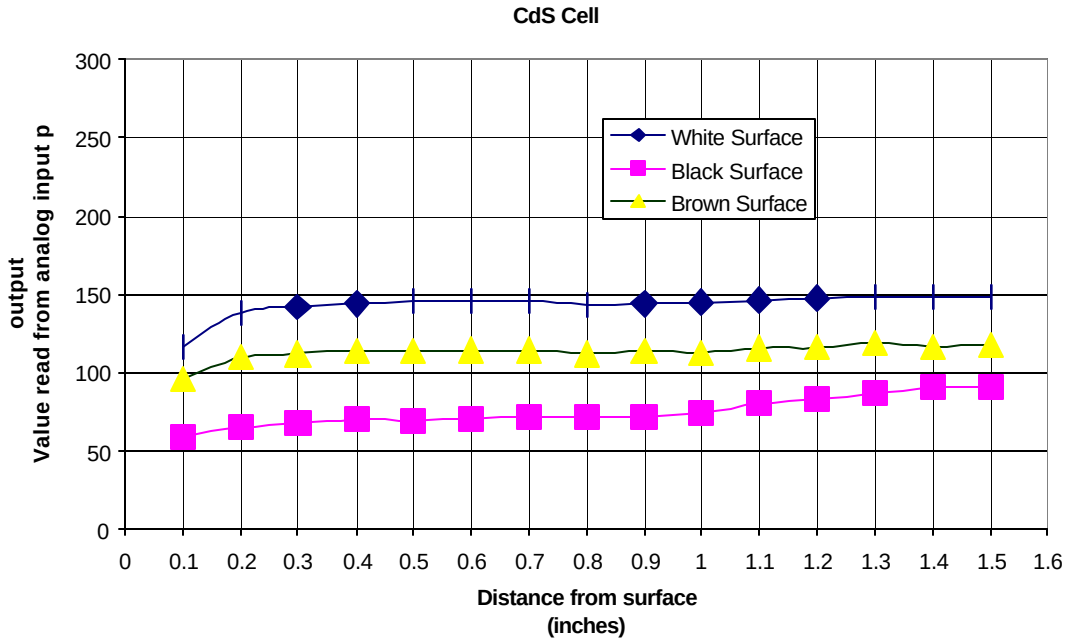


Figure 7: This graph compares the output of the CdS cell sensor over white, black and brown surfaces.

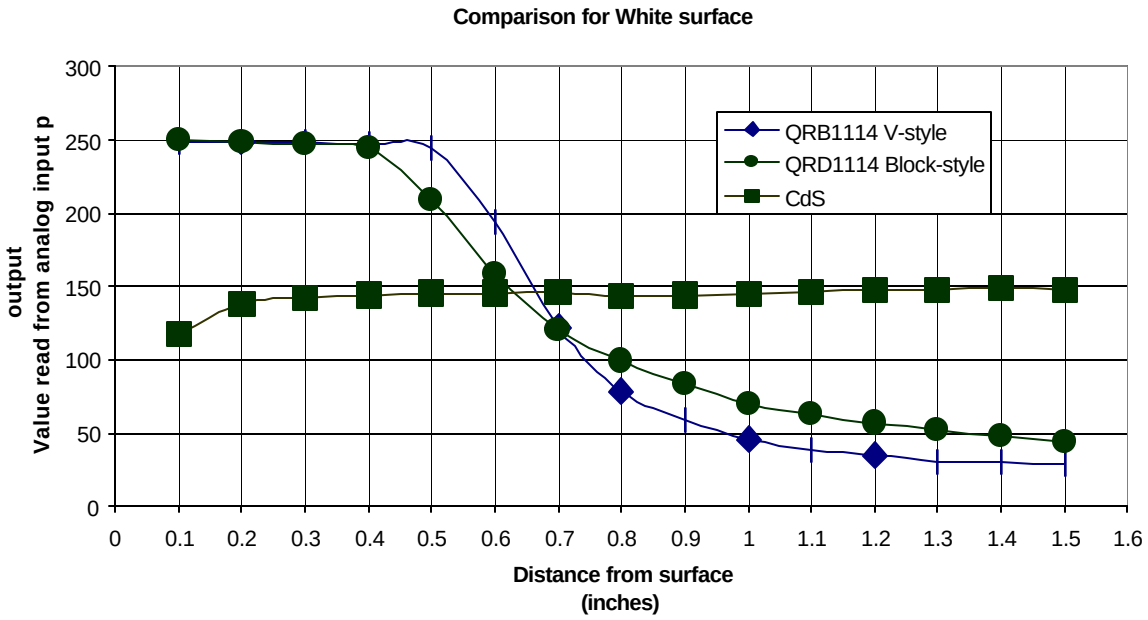


Figure 8: This graph compares the output of the CdS cell, QRB1114, QRD1114 sensors over a white background.

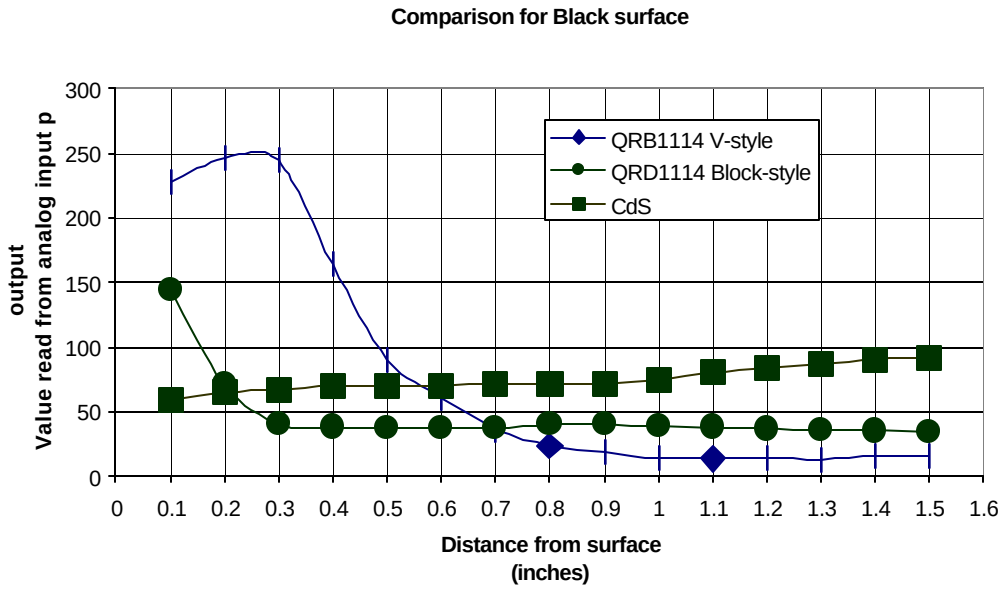


Figure 9: This graph compares the output of the CdS cell, QRB1114, QRD1114 sensors over a black background.

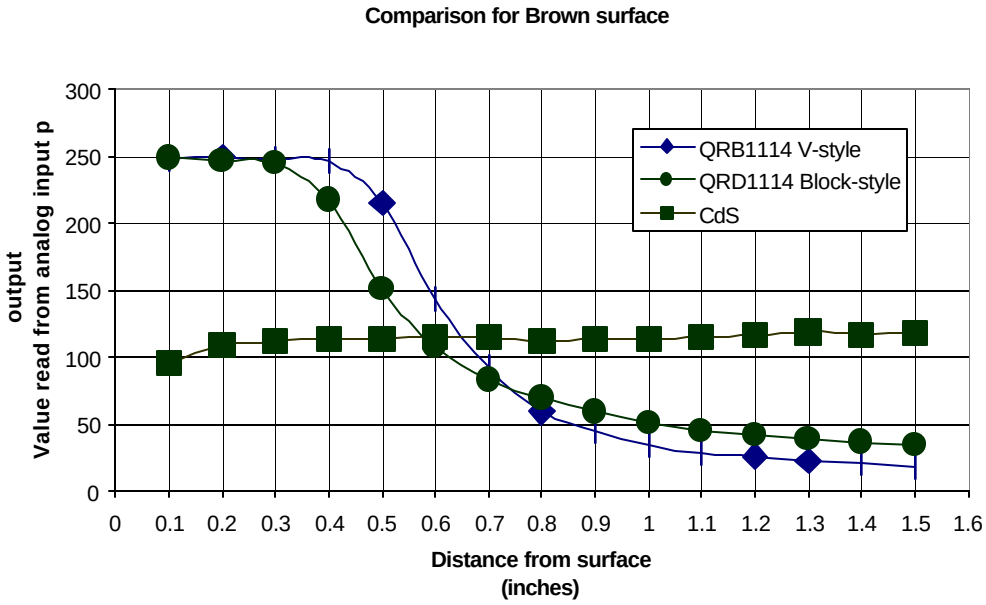


Figure 10: This graph compares the output of the CdS cell, QRB1114, QRD1114 sensors over a brown background.

Conclusions From Sensor Experiments

The plots of the results make it apparent that the photo reflectance sensors outperform the traditional CdS cell for this range and application. At the 0.5 inch range, where the sensor will be mounted, the QRB1114 outputs a value difference of nearly 160 on the analog input port over white vs. black. Similar data for the CdS show that it only has a range of approx. 70, exhibiting only half the range of the photo reflectance sensor.

Between the photo reflectance sensors, the QRB1114 has slightly more sensitivity over black, while the QRD1114 showed the greatest output distance of white vs. black surfaces. The different units of the same models also exhibited minimal discrepancy in their performance.

Due to the superior performance of the photo reflectance sensors, they were chosen and implemented to perform the line following task for the forklift. The perform

exceptionally well, being able to distinguish the line from nearly any background surface, and in vastly varied ambient lighting conditions.

Behaviors

The most elementary behavior performed by the robot is following the black tape line forming the track around the mini-factory. An obstacle avoidance behavior has been added that will stop the forklift and sound a beeper if the 40 kHz IR detector finds and obstacle in the path. This obstacle avoidance behavior is disabled when the forklift detects a load/unload station indicator.

When a load station is detected, the forklift performs a loading routine. This includes advancing until the pallet/load is detected to be in place on the forks. Then, the forks will lift the load to a predetermined height. Then the forklift will drive in reverse until it finds the line and can again perform its line following routine. A similar procedure performed during the unloading task.

Implementing the forklift loading and unloading routines was fairly simple, with my greatest difficulties lying in programming the forklift to know when to start and end the routines. The photo reflectance sensors mounted under the front at the edges are used to detect the cross in the track that indicates a load/unload station. When the station is detected and the forklift is unloaded, the load routine initiated. The forklift then lowers the forks, and line follows forward at a reduced speed until the load is detected in place on the forks. After stopping, it raises the load, and then drives slowly in reverse until the line sensors detect the line and then stops. It then drives in reverse again until the line sensors don't detect the line, and stops again. Then it rotates left until the left edge line sensor, and the center line sensor don't detect the line, stops, and then continues to rotate

left until they again see the line. This is the method used to complete the 90 degree turn. The stops between line sensing states were initially used for experimentation and trouble shooting the turning behavior. However, they proved to be necessary because when they were removed, the routine no longer functioned as well. The unloading routine is similarly executed.

These are the behaviors that have been implemented at this point. A sorting behavior could also be easily implemented using a load identification method such as color detection or bar code scanning.

Experimental Layout and Results

Experimentation was performed in the following order. First, the platform was assembled and line following behaviors were perfected. Simple line following algorithms were tried first, with more complex programs being tried later in order to achieve the best line following potential.

The next behaviors to be implemented were the forklift loading and unloading routines. Much experimentation was required to find a functioning routine for completing the 90 degree turn following the loading/unloading routine. After many trials, a best routine was found as describe above in the behavior section.

Conclusion

To my own surprise, I accomplished all of the objectives I set for the mini-forklift when I outlined the scope of the project. Wisely, I kept the objective behaviors relatively simple, because this was my first attempt at using a microcontroller or building a robot. I am very pleased at the outcome of the project. The forklift looks and performs exactly as

designed. This happy result was however obtained after many, many hours spent cursing and tweaking the hardware and software of the robot.

The most outstanding feature of the completed robot are the silicon photo transistors IR emitter detector pairs (photo reflectance sensors) used for line following. These sensors performed exceptionally well under vastly varied conditions, and saved me much trouble encountered by others using CdS cell for similar tasks. I would recommend these sensors to anyone who is interested in detecting dark and light colors. These sensors are available from Mouser Electronics with part numbers 512-QRB1114 and 512-QRD1114 for slightly more than a dollar each. The 512-QRD1114 are slightly cheaper, smaller, and seemed to perform slightly better than the other model.

If I were to start the project over, I would definitely get started into software programming earlier. I did not realize how sensitive and temperamental the software/hardware interaction would be. I would also consider using small DC motors instead of the hacked servos for the drive wheels. The Mechatronix motor control code was very problematic, and I was frustrated by the difficultly I had controlling and matching the speeds of the hacked servos. In the end, I had to use a table of preselected speed values to get the forklift to drive straight. I think more complex line following algorithms would be easier to implement (like PID control) if the speeds of the motors could be controlled more reliably.

It was beneficial for me to have and initial and final iteration of the platform. My first platform was crude, but helped me to determine the best dimensions and design for a final, neat looking platform. I would recommend this method to future robot builders.

Build a fast initial platform to work with, and then once all the needed sensors, dimensions and mechanisms have be determined, design a final platform.

Future work with the forklift will include experimenting with more complex line following routines. I have some good ideas for using PID control, but due to time constraints have not been able to experiment much with these more complex programs yet. I also would like to implement the sorting behavior. This should be easy, since I already have a photo reflectance sensor mounted on the forks that could be used to elementary color detection.

In retrospect, this class has be a incredible learning experience, exactly how other past students described it to me. The instructors and teaching assistants have been very helpful, and I'm very pleased with the outcome of my project. I'm happy to have a functioning robot that will be a reminder of my college days for many years to come.

Appendix

```
/*****************************************************************************
* Title          Forkup3                                  *
* Programmer     Chad K. Tobler                                *
* Date           April 16, 2002                           *
* Version        3                    *
*                                              *
* Description                                                  *
*   Follows line, loads and unloads cargo at designated locations.      *
*****************************************************************************/

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

#include <analog.h>
#include <clocktjp.h>
#include <servotjp.h>
#include <serialtp.h>
#include <isrdecl.h>
#include <vectors.h>
#include <stdio.h>

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

/************************* Global Variables *****************************/
int speedr, speedl;
int load=0;
int b=0;          /*for beeping and flashing*/


/************************* Function Prototypes **************************/
void get_load(void);
void unload(void);
void stop(void);
void rotate_left(void);
void rotate_right(void);
void line_follow(int set_speed_L, int set_speed_R);
void reverse(void);
void setdigport(void);
void on_off_cycle(int cycle, int output);

/***************************** Constants ********************************/
#define LEFT_MOTOR        0
#define RIGHT_MOTOR       1
#define MAX_SPEED_R   2450
#define MAX_SPEED_L   3600
#define SLOW_SPEED_R  2800
```

```
#define SLOW_SPEED_L  3360
#define REVERSE_R3343
#define REVERSE_L 2785
#define ROTATE_L   2775
#define ROTATE_R  3360
#define ZERO_SPEED   0

/************************ FEEDBACK ARRAY OUTPUT
*****************************/

#define IR_ON *(unsigned char *)(0x7000) = 0x81
#define LED1_ON *(unsigned char *)(0x7000) = 0x81
#define LED2_ON *(unsigned char *)(0x7000) = 0x82
#define LED3_ON *(unsigned char *)(0x7000) = 0x84
#define LED4_ON *(unsigned char *)(0x7000) = 0x88
#define LED5_ON *(unsigned char *)(0x7000) = 0x90
#define LOADING     *(unsigned char *)(0x7000) = 0x8F
#define UNLOADING          *(unsigned char *)(0x7000) = 0xFF

#define GOTLOAD    *(unsigned char *)(0x7000) = 0x81
#define STOP *(unsigned char *)(0x7000) = 0xFF
#define ROTATE_LEFT        *(unsigned char *)(0x7000) = 0xC0
#define ROTATE_RIGHT *(unsigned char *)(0x7000) = 0x83

#define BEEPER 0x10
#define LIGHT  0x08
#define LOADED_FLASH_CYC 350
#define REVERSE_BEEP_CYC 1400
#define BLOCKED_BEEP_CYC 1400
#define BEEPON SET_BIT(PORTD, 0x10)
#define BEEPOFF CLEAR_BIT(PORTD, 0x10)
#define LIGHTOFF CLEAR_BIT(PORTD, 0x08)
#define LIGHTON SET_BIT(PORTD, 0x08)


/************************ Fork Controls ****************************/

#define RAISEFORKS servo(2, 2500)
#define LOWERFORKS servo(2, 755)
#define MEDFORKS servo(2, 1200)




/* Enable OC4 interrupt and all servo operations */
#define SERVOS_ON     SET_BIT(TMSK1,0x10)
```

```
/*Disable OC4 interrupt: Stops all servo holding torques, useful for energy savings*/
#define SERVOS_OFF    CLEAR_BIT(TMSK1, 0x10)

/************************ Analog Channel Designations
*****************************/

#define Left    analog(2)
#define Middle  analog(3)
#define Right   analog(7)

#define INPUT    analog(0)
#define IRCAN    analog(5)
#define LWING    analog(6)
#define RWING    analog(4)
#define LOAD     analog(1)

/************************ Sensor Thresholds *****************************/

#define LINE_THRESHOLD 100
#define IR_THRESHOLD   110



/******************************** Main ***********************************/

void main(void)
{

  int last_left, last_right;


  init_analog();
  init_clocktjp();
  init_servotjp();
  setdigport();
  IR_ON;


  while(1)
  {
        line_follow(MAX_SPEED_L, MAX_SPEED_R);

        if(IRCAN>IR_THRESHOLD)
                {
                        BEEPON;
```

```
                stop();
                BEEPOFF;
                wait(500);
        }


    if(LWING<LINE_THRESHOLD || RWING<LINE_THRESHOLD)
    {
        stop();

        if(load==0)
        {
            get_load();
        }
        else if(load==1)
            unload();
        /*else if(LWING<LINE_THRESHOLD)
        rotate_left();
    else
        rotate_right();*/
    }


  /* if(Right>LINE_THRESHOLD && Left>LINE_THRESHOLD &&
Middle<LINE_THRESHOLD)
   {
    if (last_left<last_right)
        {
            speedr=MAX_SPEED_R;
                speedl=SLOW_SPEED_L;
                LED4_ON;
                wait(30);
            }
            else
            {
                speedr=SLOW_SPEED_R;
                speedl=MAX_SPEED_L;
                LED5_ON;
                wait(30);
            }
    }

    last_left=Left;
    last_right=Right;  */

    if(load)
```

```
        {
                on_off_cycle(LOADED_FLASH_CYC, LIGHT);
   }

        servo(RIGHT_MOTOR, speedr);
   servo(LEFT_MOTOR, speedl);

  }     /*end of while loop*/

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

void get_load(void)
{
        LOADING;
        LOWERFORKS;

        while(load==0)
        {
                if(INPUT>100 || LOAD>200)
                {
                        load=1;
                        LIGHTON;
                }
                else
                load=0;

                line_follow(SLOW_SPEED_L, SLOW_SPEED_R);

                servo(RIGHT_MOTOR, speedr);
        servo(LEFT_MOTOR, speedl);
        }

        GOTLOAD;
        stop();

        RAISEFORKS;
        wait(1000);

        while(LWING>LINE_THRESHOLD)
        {
                reverse();
   }
   stop();
   while(LWING<LINE_THRESHOLD)
        {
```

```
                reverse();
    }

    BEEPOFF;
    stop();


    rotate_left();

    /*if(LWING<LINE_THRESHOLD)
        rotate_left();
    else if(RWING<LINE_THRESHOLD)
        rotate_left();   */

}

void unload(void)
{
        UNLOADING;
        LIGHTON;
                        /*Drive forward to the end of the line before unloading*/
        while(Left<LINE_THRESHOLD || Middle<LINE_THRESHOLD ||
Right<LINE_THRESHOLD)
        {
                line_follow(SLOW_SPEED_L, SLOW_SPEED_R);

                servo(RIGHT_MOTOR, speedr);
        servo(LEFT_MOTOR, speedl);
        }

        stop();

        LOWERFORKS;
        wait(1000);

        while(LWING>LINE_THRESHOLD)
        {
                reverse();
    }
    stop();
    while(LWING<LINE_THRESHOLD)
        {
                reverse();
    }

    if(INPUT>100 || LOAD>200)
```

```
                load=1;
                else
                load=0;

   LIGHTOFF;
   BEEPOFF;
   stop();


   rotate_left();

}

void stop(void)
 {
        speedl=ZERO_SPEED;
        speedr=ZERO_SPEED;
        servo(RIGHT_MOTOR, speedr);
   servo(LEFT_MOTOR, speedl);
        wait(700);
 }
void rotate_left(void)
 {
        ROTATE_LEFT;
        while(LWING<LINE_THRESHOLD || Middle<LINE_THRESHOLD)
        {
                speedl=ROTATE_L;
                speedr=ROTATE_L;
                servo(RIGHT_MOTOR, speedr);
        servo(LEFT_MOTOR, speedl);

   }
   stop();
   while(Middle>LINE_THRESHOLD)
        {
                speedl=ROTATE_L;
                speedr=ROTATE_L;
                servo(RIGHT_MOTOR, speedr);
        servo(LEFT_MOTOR, speedl);
        /*wait(ttimel);*/

   }
   stop();
 }

void rotate_right(void)
```

```
{
        ROTATE_RIGHT;
        while(RWING<LINE_THRESHOLD)
        {
                speedl=ROTATE_R;
                speedr=ROTATE_R;
                servo(RIGHT_MOTOR, speedr);
        servo(LEFT_MOTOR, speedl);

   }
   while(RWING>LINE_THRESHOLD)
        {
                speedl=ROTATE_R;
                speedr=ROTATE_R;
                servo(RIGHT_MOTOR, speedr);
        servo(LEFT_MOTOR, speedl);
        /*wait(ttimer);*/

   }
   stop();
 }

void line_follow(int set_speed_L, int set_speed_R)
{
        if(Left<Middle && Left<Right)
                {
                        speedr=set_speed_R;
                        speedl=ZERO_SPEED;
                        LED1_ON;
                }
        else if(Right<Middle && Right<Left)
                {
                        speedr=ZERO_SPEED;
                        speedl=set_speed_L;
                        LED3_ON;
                }
                else
                        {
                                speedr=set_speed_R;
                                speedl=set_speed_L;
                                LED2_ON;
                        }
}

void reverse(void)
{
```

```
        on_off_cycle(REVERSE_BEEP_CYC, BEEPER);

    /*if(Left<Middle && Left<Right)
                {
                        speedr=REVERSE_R;
                        speedl=ZERO_SPEED;
                        LED1_ON;
                }
        else if(Right<Middle && Right<Left)
                {
                        speedr=ZERO_SPEED;
                        speedl=REVERSE_L;
                        LED3_ON;
                }
                else
                        {
                                speedr=REVERSE_R;
                                speedl=REVERSE_L;
                                LED2_ON;
                        }*/

                speedr=REVERSE_R;
                speedl=REVERSE_L;

                servo(RIGHT_MOTOR, speedr);
        servo(LEFT_MOTOR, speedl);

}

/* Obtained from Boris Yaw who obtained it from Cynthia Manya*/
void setdigport(void)
{
        SET_BIT(DDRD, 0x04);            /*set PortD bits 2,3,4 as digital outputs*/
        SET_BIT(DDRD, 0x08);
        SET_BIT(DDRD, 0x10);
        CLEAR_BIT(PORTD, 0x04);
        CLEAR_BIT(PORTD, 0x08);
        CLEAR_BIT(PORTD, 0x10);
}

void on_off_cycle(int cycle, int output)
{
        int mod, quot;
```

```
        mod=b%cycle;
        quot=cycle/2;

        if((mod) < (quot))
                SET_BIT(PORTD, output);
        if((mod) > (quot))
                CLEAR_BIT(PORTD, output);

        if(b<(cycle*10))
                b++;
   else
                b=0;
}
```