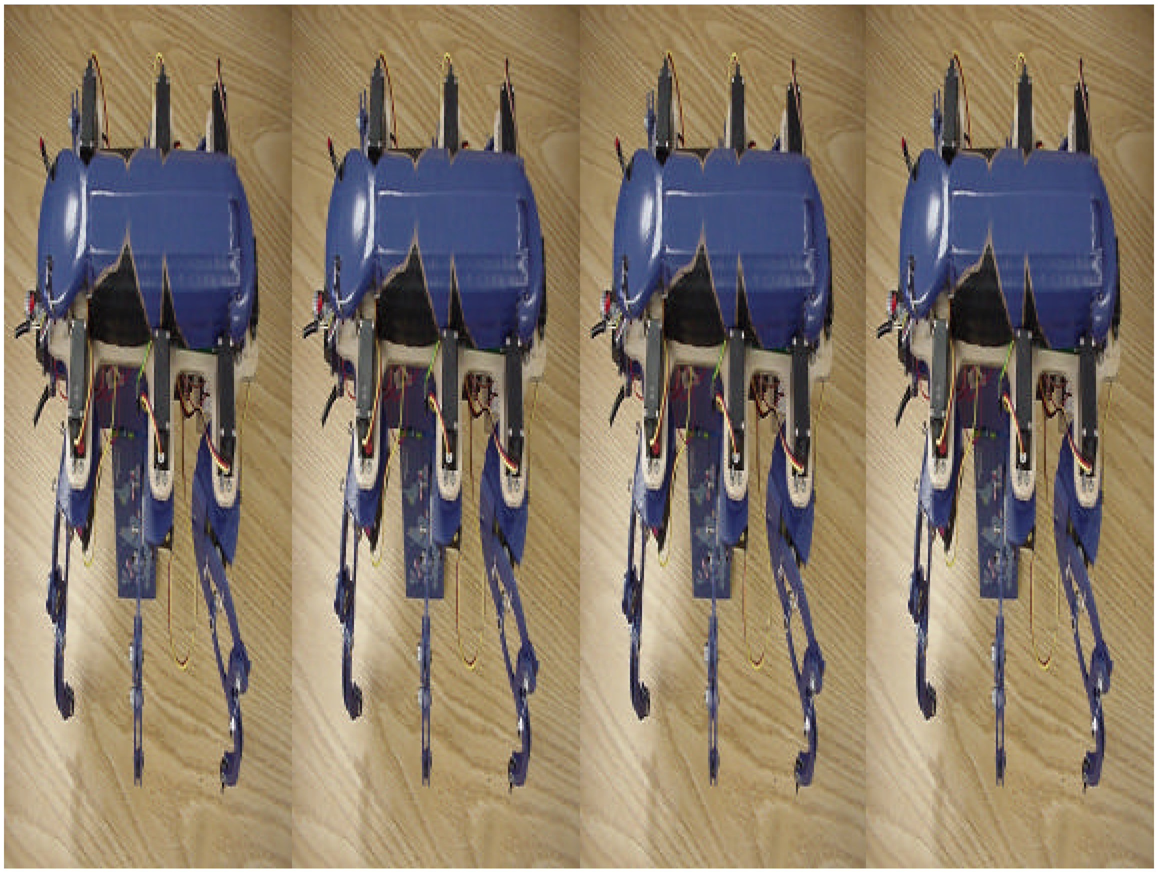

Spider Walker
The Six-Legged Insect

Garew E. Walker

April 23, 2002

University of Florida
Intelligent Machines Design Laboratory
EEL 5666

TABLE OF CONTENTS

ABSTRACT

This paper presents the ideas and work behind the development of the six-legged robot Spider Walker. Spider Walker was partly inspired by Robobug and borrows some ideas from Marc Poe's Hannibal and Cynthia Manya's Kisa robot. Many other research robots, as well as living insects, served as inspiration for design and behavioral characteristics. Spider Walker uses IR and CdS cell sensors to interact with its environment. It uses a total of 13 servos for all leg and special mechanism actuation. It uses light-emitting diodes (LED's) and a piezo mini buzzer for communication with the outside world. Finally, for its intelligence, it uses both a TJPro11 micro-controller with the Motorola MC68HC11A1 processor and a MSCC11 micro-controller with the MC68HC811E2 processor.

The main proposed behaviors of Spider Walker were to seek out a light source, avoid any walls or low-lying obstacles in its path, and crawl under any overhanging objects in its way. Once the light was found, Spider Walker was then to deliver any objects loaded onto its carrier. All proposed objectives for this robot project were met with additional features incorporated as well.

Executive Summary

Spider Walker successfully completed all the tasks that I initially intended it to and more. It first begins by standing up in its upright position for a few seconds while things warm up. It then begins to walk, using a tripod gait, and heads towards a light source serving as a beacon. I purposefully place obstacles in the robot's path and so that it can demonstrate it's behaviors. Whenever Spider Walker encounters a wall or low-lying object in its path, it does a special beep pattern to indicate that there is an object blocking it its way. It then sets up itself in standing position, walks backwards for a little while, and then turns to the left to continue walking forward. This is repeated until its path is clear.

As Spider Walker turns to the left or right to head towards the light, it flashes it left or right LED's, respectively, to give feedback at to what direction it's supposed to be going in. When it's walking straight, the LED's remain on but do not flash. Whenever Spider Walker encounters an overhanging object that's high enough for it to craw underneath, it lowers itself, crawls under it, stands up and continues heading towards the light. When it finally reaches the light, it orients itself with it, turns around and drops off two small objects loaded onto it.

Introduction

I decided to build a walking robot mainly because I thought it was so cool. It seemed like more of a challenge than the typical wheeled robots I saw in previous IMDL classes, and I figured that I would end up learning much more about robot design problems and ways to fix them. Developing Spider Walker also helped me learn about something I never anticipated needing, the Serial Communication Interface. Since I have a Mechanical Engineering background, I finished most of the design stuff within the first five to six weeks but then got stuck for about a month just learning how to get my two boards to communicate via the SCI. Not much electronic or software progress could be made without first figuring out this part.

Once getting my boards to communicate with each other, I started developing code and doing all the walking gait experiments. The robot uses 3 Infra Red sensors and 3 CdS cells for sensing its environment, a piezo buzzer and LED's for communication with outsiders, a total of 13 servos for all its actuation, and both the TJPro11 and MSCC11 boards for smarts.

This project consumed countless amount of hours to complete, but kept me entranced and forced me to learn a tremendous amount of worthwhile "stuff."

Integrated System

Spider Walker uses both the TJPro11 and the MSCC11 micro-controller boards. Figure 1 gives a top view of the robot with boards in view. The TJPro11 board has a Motorola

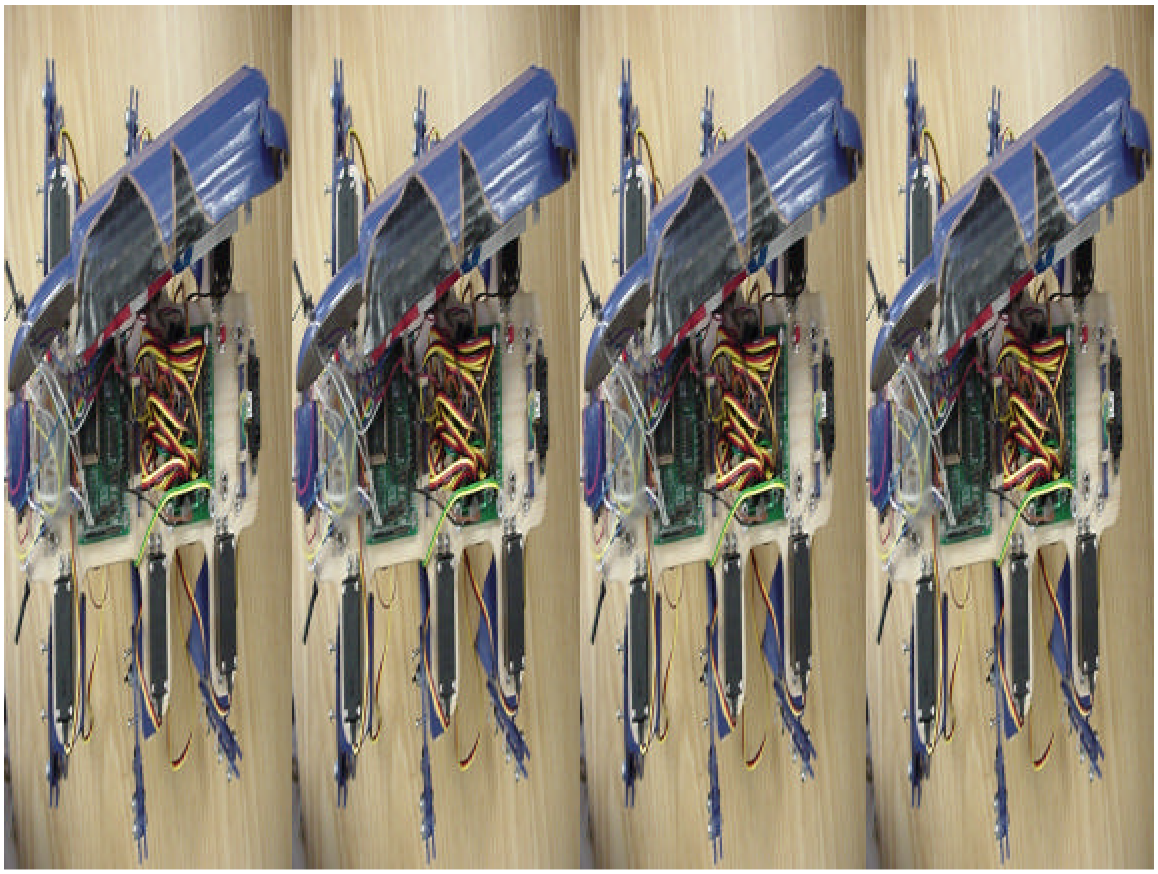

Figure 1. **Controller board setup as viewed from the top of the robot. The TJPro11 board ("brain board") resides to the left, while the MSCC11 board ("servo board") is the one to the right with all servo cables hooked up to it .**

MC68HC11A1 microprocessor and 32 KSRAM. A close up view of this board is shown in figure 2 and shows all the sensor, feedback, mode switch, and reset switch wiring connected to it. Since it has a considerable amount of memory, it was used to do all the thinking and processing of sensor data, and then tell the MSCC11 board what to do. The

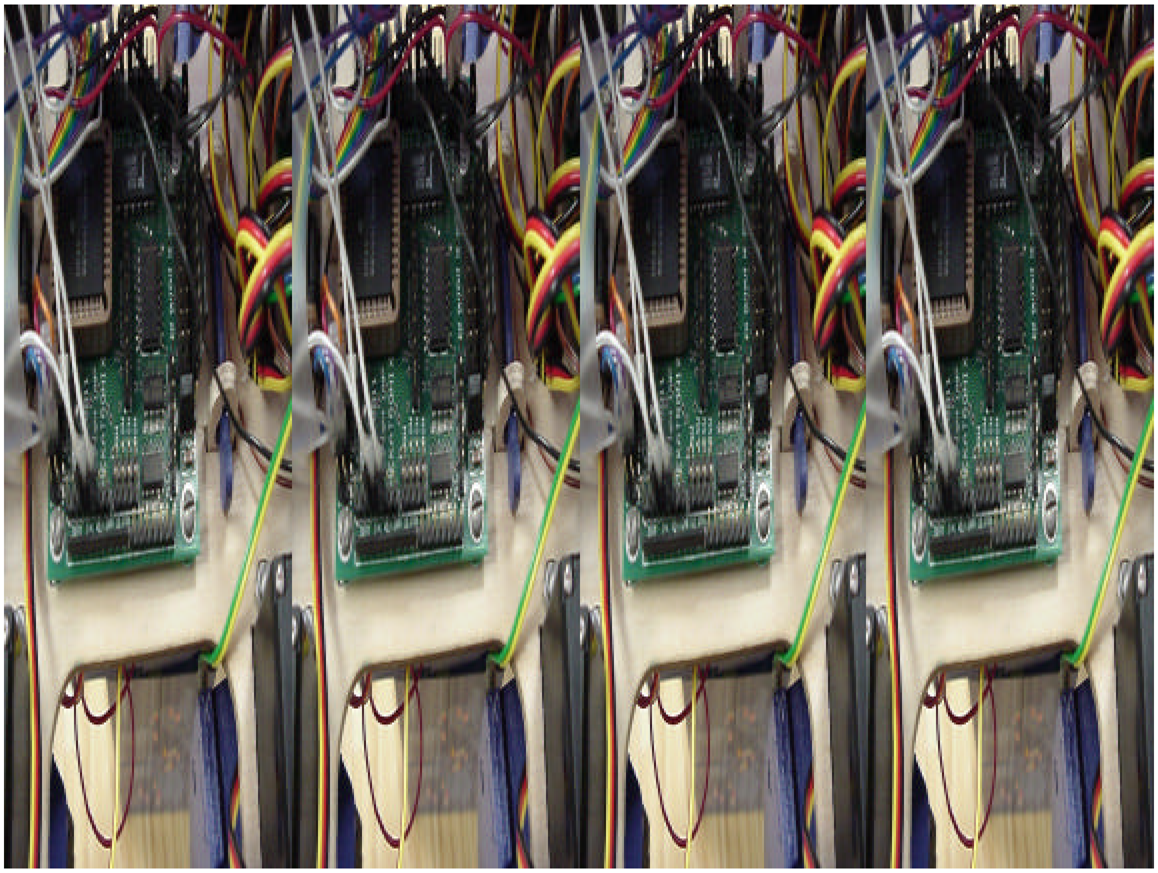

Figure 2. **TJPro11 "brain board" with sensor, buzzer, LED, mode switch, and reset switch connected.**

MSCC11 board, which controls all the servos, is shown in figure 3 and has a MC68HC811E2 processor with 2 K EEPROM and just accepts commands from the TJPro11 board through the SCI port. The physical connection is done by just using a two wire cable to connect the transmit pin from the first board to the receive pin on the second board, and connect the receive pin from the first board to the transmit pin on the second board. The boards must also share a common ground.

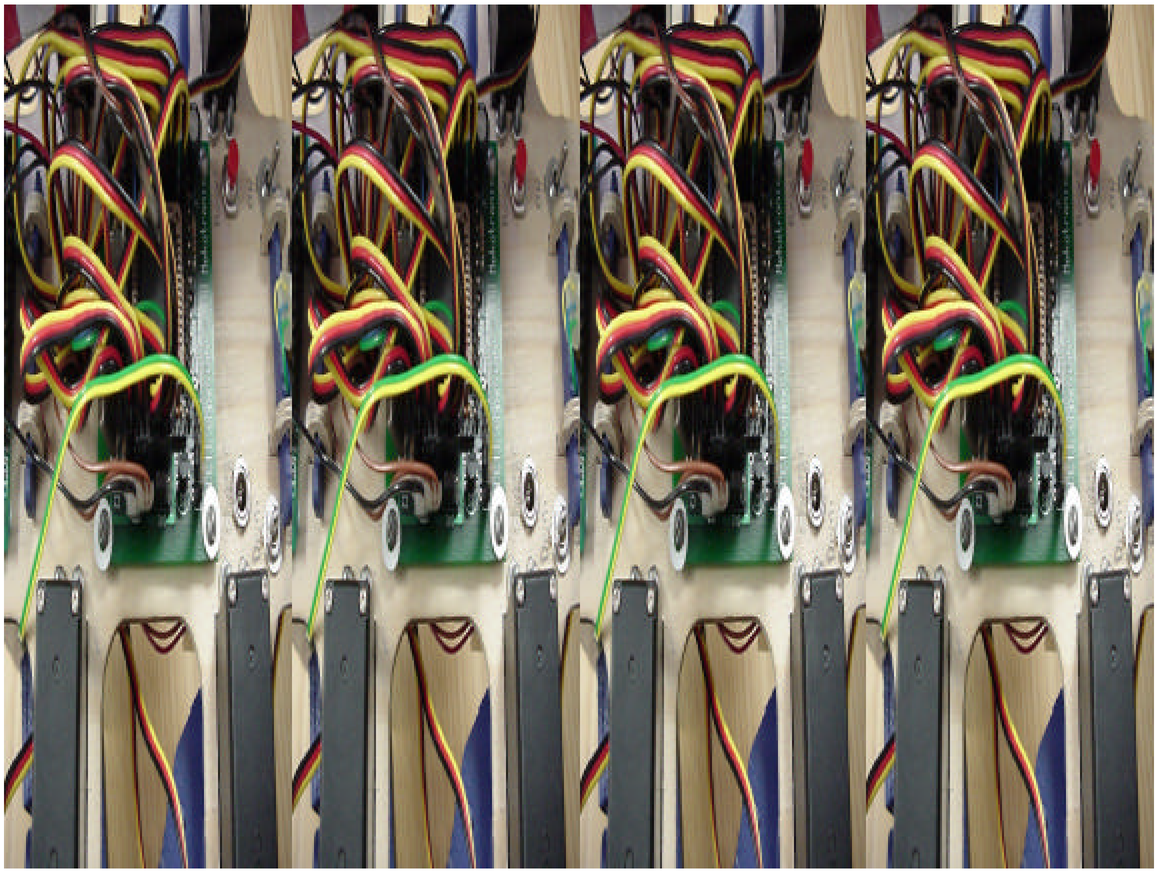

Figure 3. **TMSCC11 "servo board" with a serial connection to the TJPro11 board, it's own reset and mode switches, and all 13 servo cable connections (ground, poser, and signal wires for each cable). Also seen in this photo is the power on/off switch in the upper right hand corner, and the charge socket directly below the red reset switch.**

Spider Walker's platform was designed to accommodate the two micro-controller boards, sensors, batteries, and servos. The sensors help the robot interact with its environment and perform its behaviors. Figure 4 shows a block diagram of the main components of Spider Walker and how they are connected:

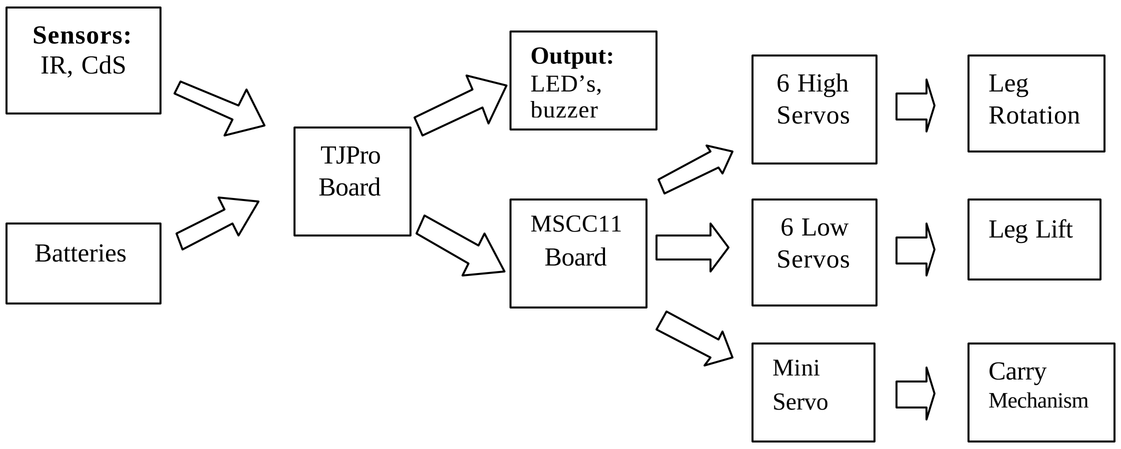

Figure 4. **Block diagram of the Spider Walker's integrated system. This illustrates the basic sensor, feedback, power, controller, and actuator connections.**

The diagram shows the different uses for the six upper and lower servos as well as the mini servo. There are two separate 6-pack AA rechargeable NiCad battery cases providing power. One pack provides power for all the electronics while the other powers to the servos.

Mobile Platform

Body

The main body of this robot is made out of model airplane board. This allows the frame to remain relatively lightweight but strong enough to withstand bending torques. This is also because of the relative ease of designing parts in AutoCAD and having the Machine

Intelligence Lab (MIL) T-tech machine cut them out on this type of material.

The body section has a basic rectangular form and has three rectangular holes cut out along both of its long sides. These holes accommodate the six servos that provide the backward/forward swinging motion of the legs. Smaller circular holes were cut through this section for placement of the switches and charger socket. Figure 5 illustrates the AutoCAD drawing of this platform.

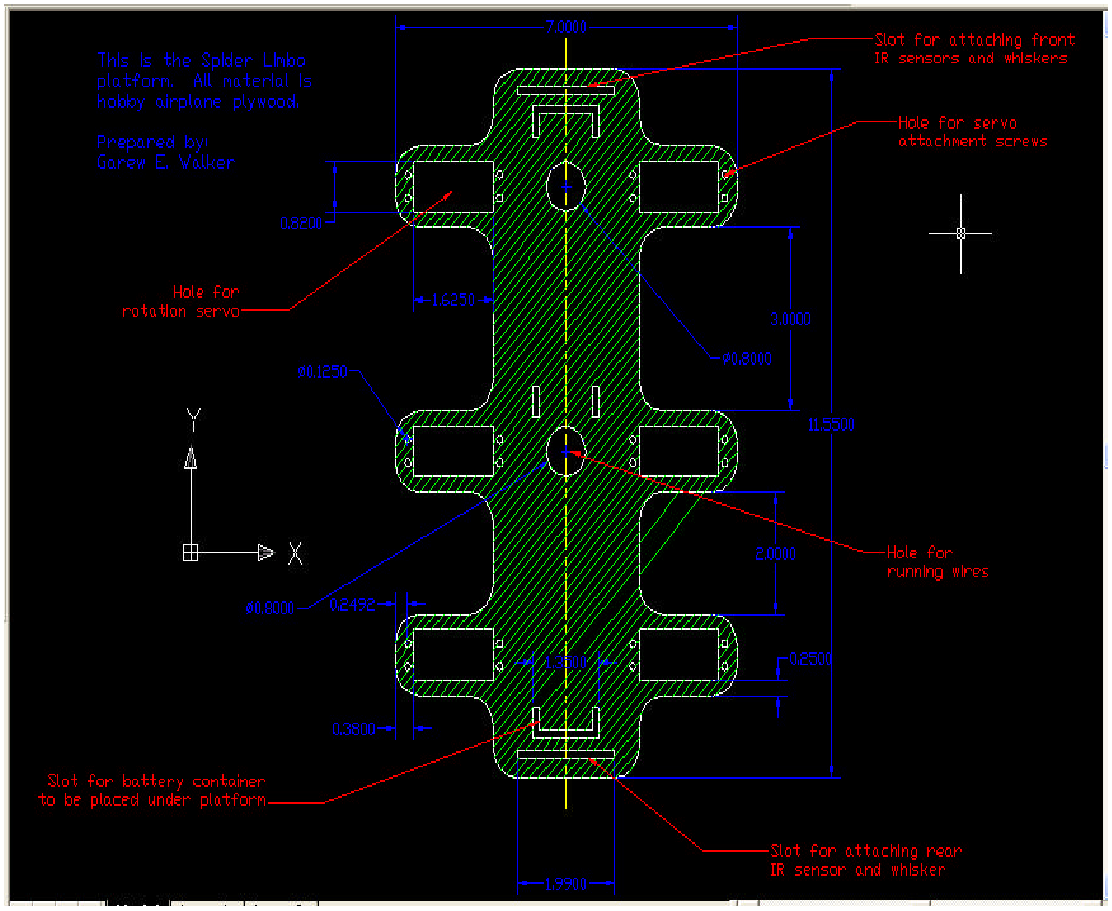
This is the Spider Limbo platform. All material is hobby airplane plywood.
Prepared by: Garew E. Walker
7.0000
Slot for attaching front IR sensors and whiskers
Hole for servo attachment screws
0.8200
Hole for rotation servo
1.6250
3.0000
ø0.8000
ø0.1250
11.5500
Y
X
Hole for running wires
2.0000
ø0.8000
0.2492
1.3500
0.2500
0.3800
Slot for battery container to be placed under platform
Slot for attaching rear IR sensor and whisker
1.9900

Figure 5. **AutoCAD rendering of Spider Walker platform design. This illustration show s the holes for fasteners as well as dimensions.**

The battery case was designed to minimize space, weight, and provide a location for the carrier to be placed on the underside of the robot. Figure 6 shows what the parts of the case look like in my AutoCAD design, while figure 7 is a photograph of the actual assembled case.

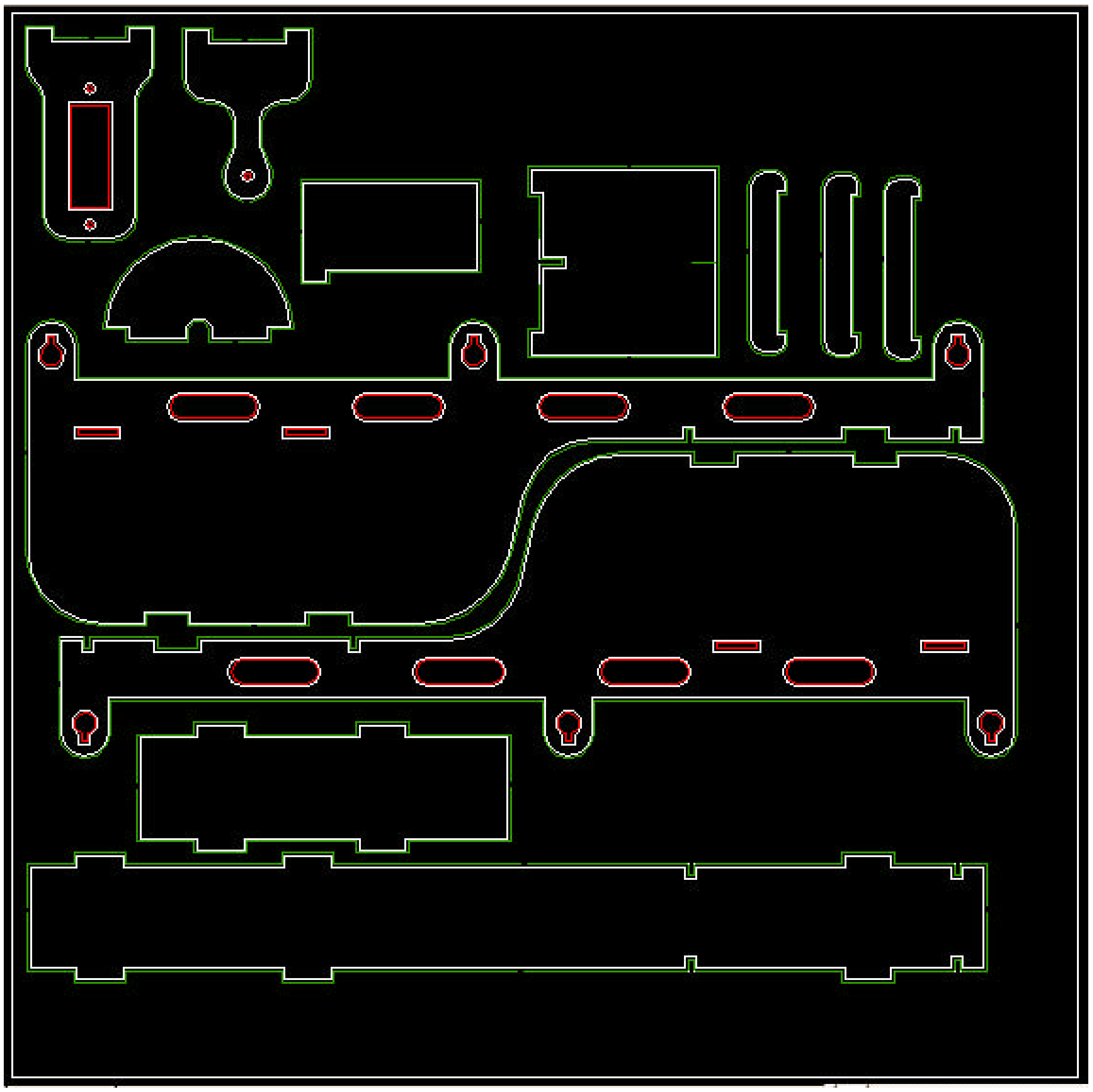

Figure 6. **AutoCAD rendering of Spider Walker battery case and object carrier design. These are the unassembled parts.**

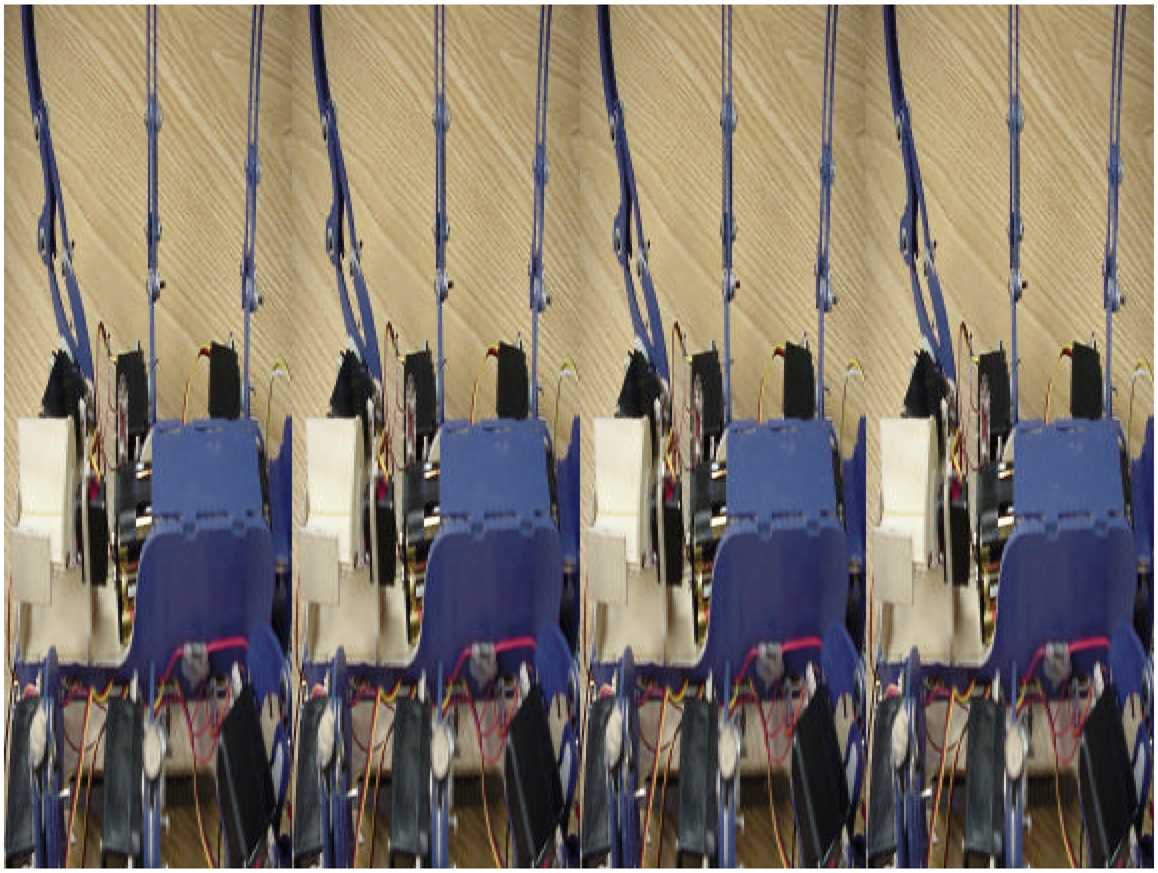

Figure 7. **This is a photograph of the assembled battery case and carrier mechanism.**

As can be seen, I designed the case so that both battery packs are located in the front portion of the robot, while the carrier mechanism is at the rear. Also, the battery holder at the front is not closed, but allows the battery cases to be slid back and forth as desired. I did this so that I could have some control over where the robot's center of gravity was located. If I had problems with programming a balanced gait, I could simply increase the weight of the carried objects or, better yet, slide the battery packs backwards or forwards to shift the center of gravity more towards the rear or the front, respectively.

I came up with the idea for Spider Walker's top cover after the robot was completely built and functioning. This served more of a decorative purpose, and it also helped to protect the electronics and wiring from damage.

Legs

The six legs were designed to minimize servo torque requirements. The rotating links (links 2 and 3 below) are long enough so that there is at least enough variation in body height when standing and crawling so that the robot can both clear the overhang and still be able to balance itself as it crawls underneath.

Each leg is essentially a four-bar mechanism. This means that each has four links connected by joints that allow it to move up and down as shown in figure 8.

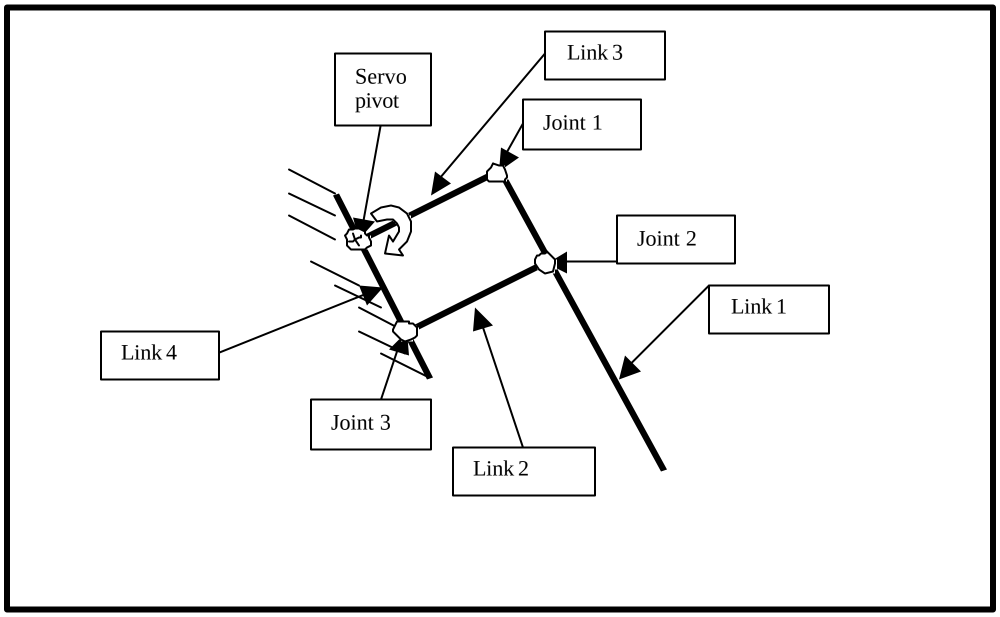

Figure 8. **Leg mechanism depicting the necessary links and the pivot pint for the leg lo wering/raising servo.**

The legs are made of the same material as the body. Figure 9 shows a photograph of the actual legs as mounted on the robot.

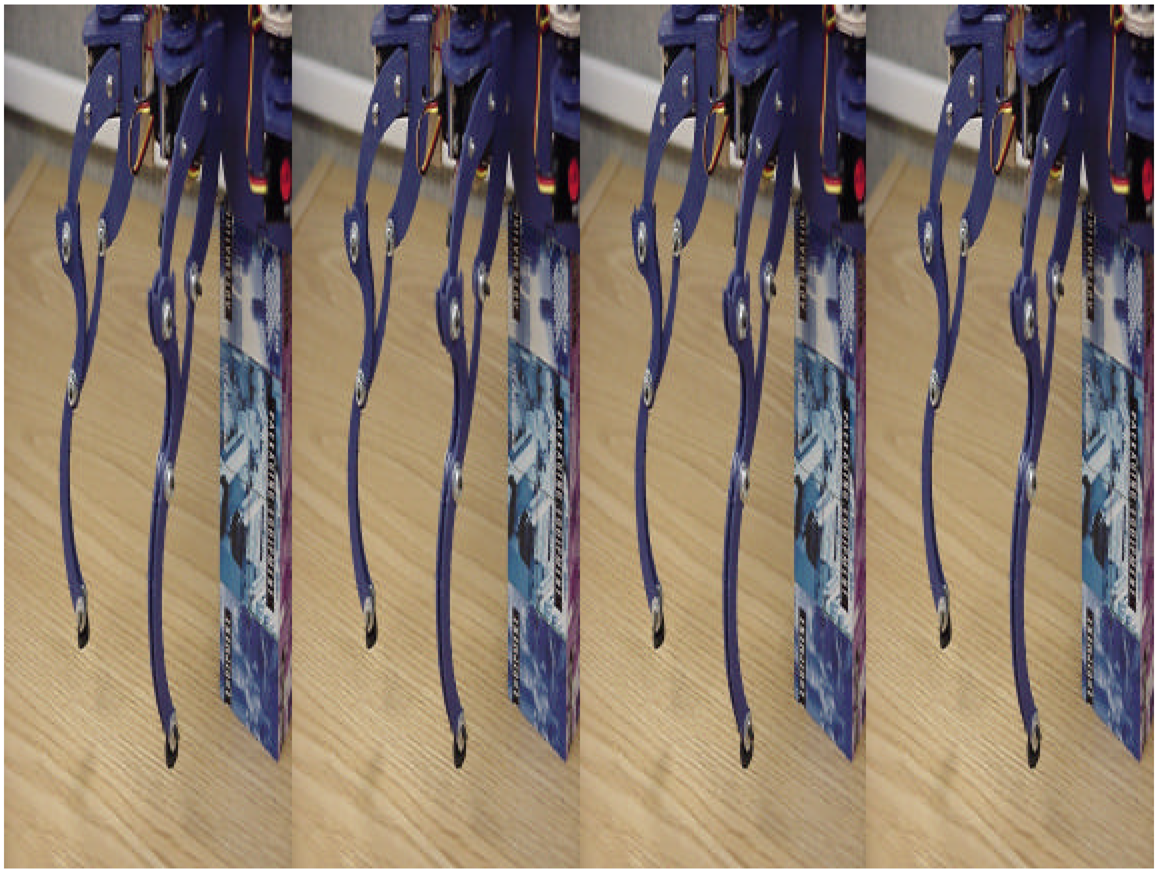

Figure 9. **Photograph of physical legs after they were finally constructed, assembled, and painted**

<u>Actuation</u>

Servomotors do all of Spider Walker's actuation. As mentioned above, there are six servos to perform the forward/backward (swing) motion of the legs. There are also six other servos connected to each leg as shown in figure 8 that will control the upward/downward motion of each leg. The required torque for each servo is a function of the robots weight and leg design. Basically, the heavier the robot or the longer links 2 and 3 as shown in figure 8, the more torque needed to keep the robot standing. I used standard Hitec HS-300 42 oz-in servos for all leg motion and a Hitec HS-81 Micro 32 oz-in servo for the carrier mechanism as shown in figure 10.

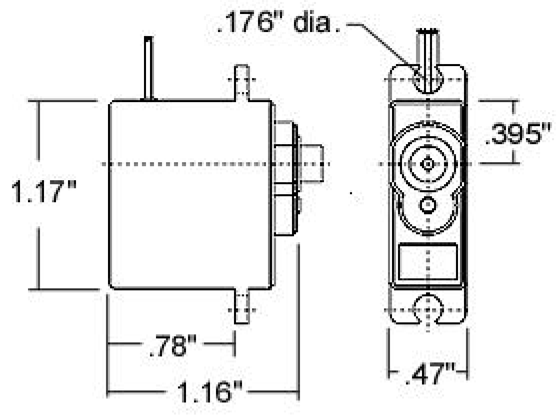

Figure 10. **Drawing of Hitec HS-81 micro servo with dimensions to illustrate actual size. This servo was used for actuation of the carrier mechanism and was chosen for it small size.**

The carrier mechanism is relatively simple in design and its basic setup is depicted in figure 11 below. The object rest holds the object and pivots about the axis connecting the servo horn center and the outside end pivot point. The object rest rotates by 90 degrees to drop off the first object and then rotates another 90 degrees in order to drop the second object.

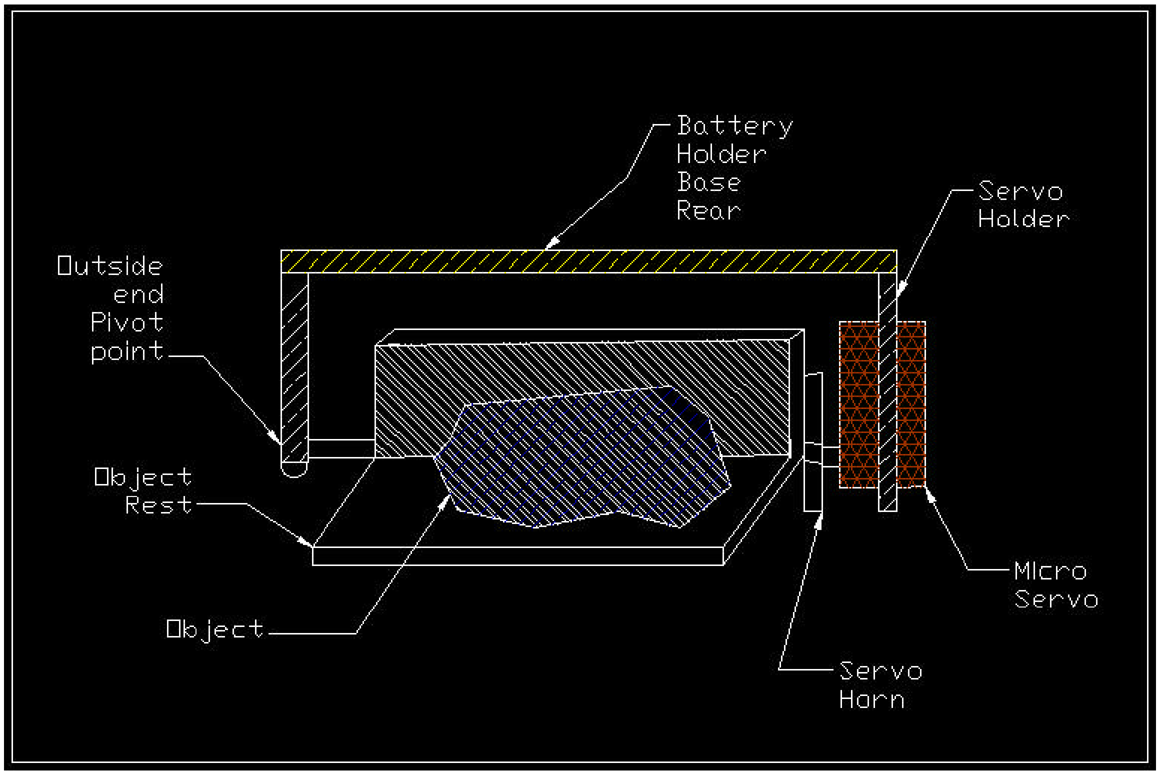

Figure 11. **Carrier mechanism design with labels of specific components and the object placement area.**

The 42 oz-in servos actually provide up to 49 oz-in at 6 VDC and therefore allow Spider Walker to have a comfortable amount of torque to keep itself standing. I was originally worried about needing to use higher torque servos for lifting the legs, but I found that these work just fine, especially since I chose to use AA batteries instead of the larger C cells. Heavier walking robots sometimes have a hard time standing. In such a case, it's probably best to just go ahead and buy higher torque lifting servos to provide enough power.

Sensors

Spider Walker has a total of 6 sensors. There are 3 Infra Red (IR) emitter/detector pair sensors and 3 Cadmium Sulfide (CdS) photoresistor sensors. Two of the IR sensors reside at the front of the robot and combine to serve as one unique sensor. The third IR sensor is located in Spider Walker's rear and is used to detect when the robot has completely cleared an overhang. The CdS sensors are all located in the front of the robot. One points to the left, the other points directly forward, and the third points to the right.

The combination of the two IR sensors in front, which I also call the "height detection sensor", is designed to help the robot implement two of its main behaviors, crawling and obstacle avoidance. The main behavior of interest is the ability to crawl under overhanging objects by detecting whether there is an object in front of the robot while simultaneously determining whether the robot can crawl underneath the object.

The height detection sensor is simple in concept and consists of two GP2D12's (one of which is shown in figure 12 below). I placed these sensors on a specially designed piece of plywood slab positioned in front of Spider Walker's body and perpendicular to it. So, the whole body and sensor setup ends up looking like the letter "T" when viewed from the robot's side. This can be seen in figure 13 of page 19 below.

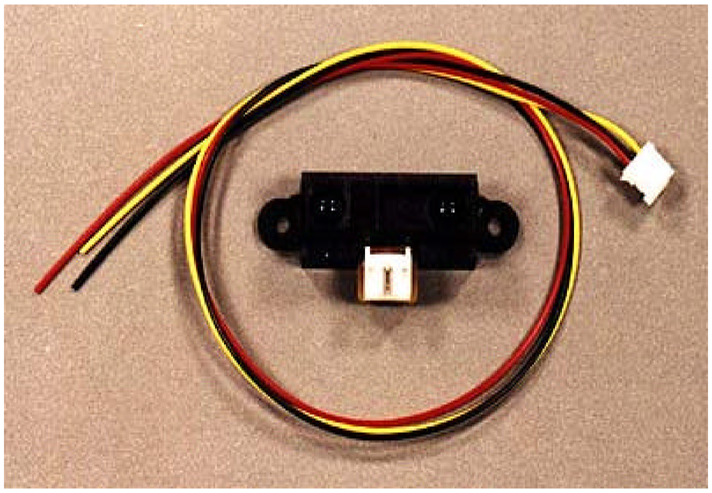

Figure 12. **Picture of one of the Sharp GP2D12 Infrared Ranger's used in the height detection sensor.**

This height detection sensor works by using the GP2D12's to send out and detect IR signals and determine if there is an object in its forward proximity. With the two sensors taking readings simultaneously, the processor can determine if there is an object within the threshold of concern for either sensor. If the top sensor detects an object and the bottom sensor does not, then it is safe to say that the robot has encountered an overhang under which it can crawl. Of course, there must be no objects too low for the bottom sensor to detect, or they will not be noted. A pair of whiskers on the bottom of the two front legs could remedy this problem.

In the other case, if the bottom sensor ever observes an object, then it is safe to say that there is an object (i.e. a wall) in front of the robot to be avoided. Normal obstacle avoidance would take over in this case.

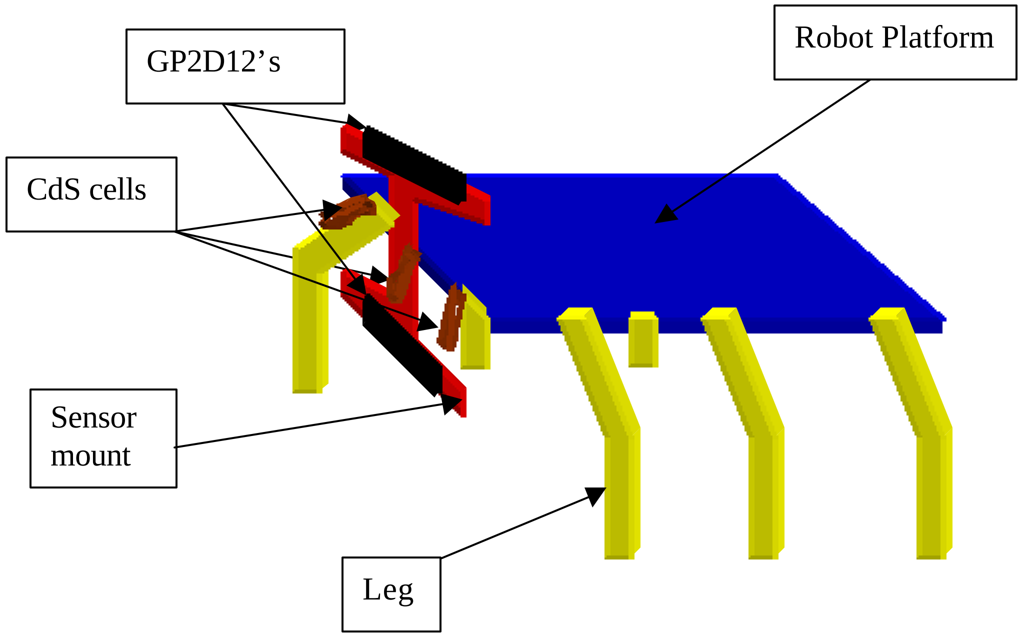

Figure 13. **Schematic of height detection sensor mounted onto the robot platform.**

Both GP2D12's were connected to two analog pins on port E of the TJPro11 board.

As shown in figure 13, the 3 CdS cells were placed in front of the robot and allowed the robot to locate the light source by turning towards the CdS cell with the lowest reading. These CdS cells are basically resistors that vary in value based on the amount of incident light intensity. This means that a change in the amount of light falling on the cell changes the value of the cells output resistance, which is a readable analog value. The Analog to Digital (A/D) port E on the TJPro11 board can then convert this value into a digital value that the microprocessor can understand.

The port E pin connections for each CdS cell is shown below:

CdS_left	=	PE5
CdS_middle	=	PE4
CdS_right	=	PE6

I enclosed each CdS cell with 1-inch long tubes of 3/16-inch diameter black heat shrink. I also placed hot glue at the rear of each tube and clamped it off so that light coming in from the back would not interfere with the sensor readings. Enclosing the cells in tubes allowed them to pick up much less light from sources to the side. Figures 14 and 15 show pictures of both the IR and CdS sensor setups.

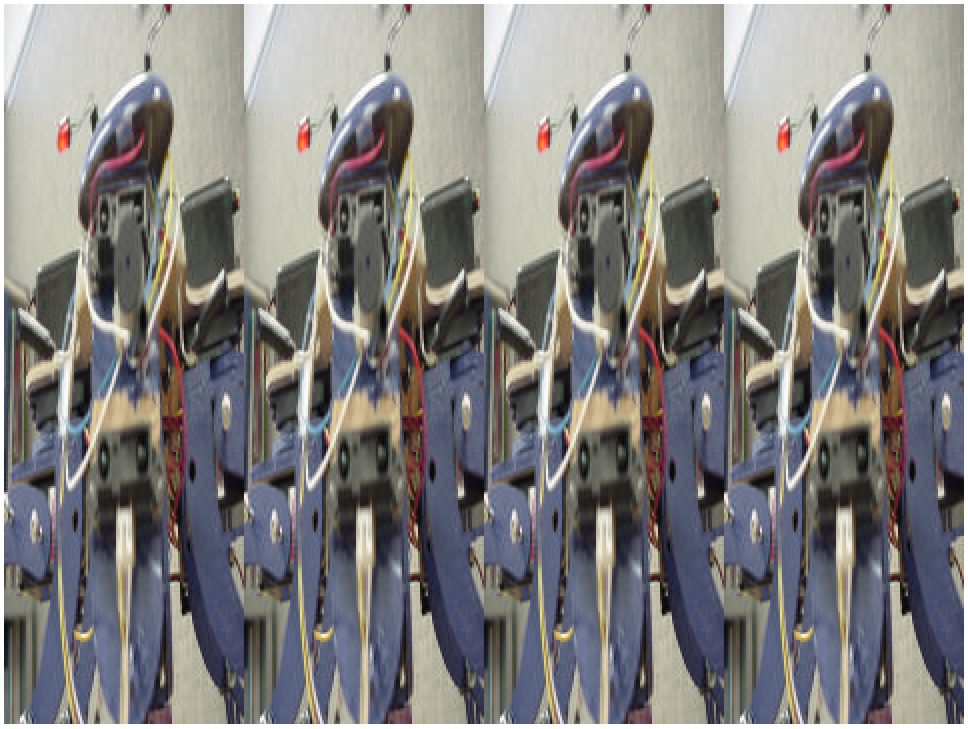

Figure 14. **Photo of IR and CdS sensor setup as seen from the lower front on the robot.. This photograph gives a depiction of the height detection sensor suing the two GP2D12's.**

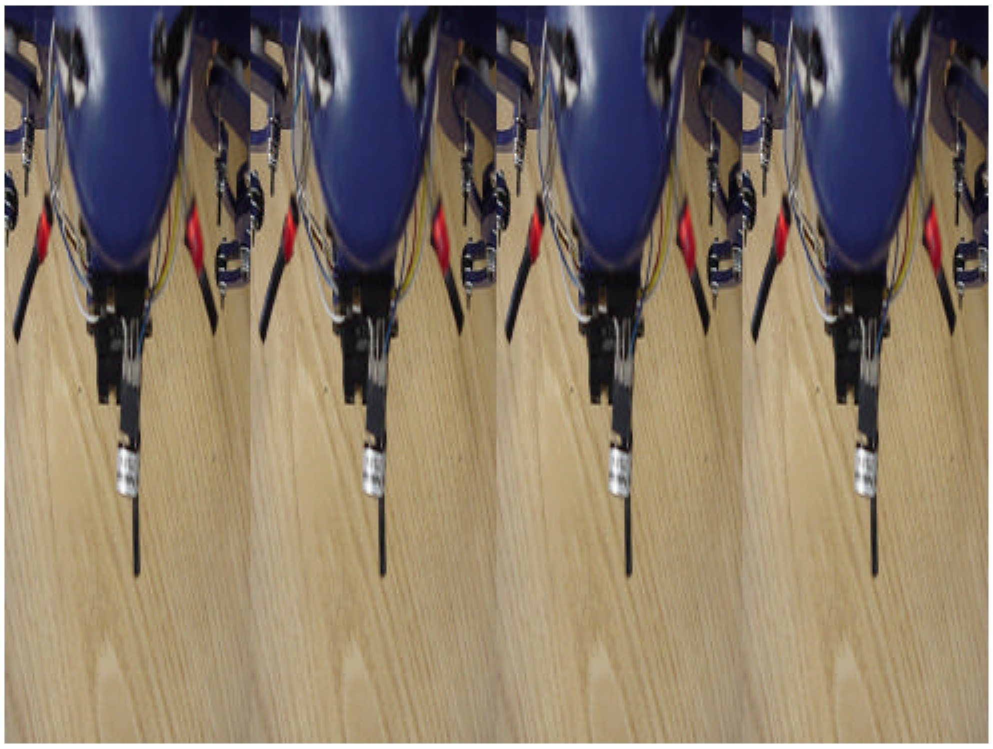

Figure 15. Photo of CdS sensor setup as seen from the top of the robot.. This photograph shows how the CdS sensors were arranged from left to right.

Behaviors

Using the sensors and actuators described above, Spider Walker is able to search for a light source that serves as a beacon. The carrier mechanism is first loaded with two small objects. The robot is then left on its own to find the light and deliver the objects to that location. Whenever the robot encounters a wall or any low-laying object to be avoided, it beeps to signal that there is something in its way that it must avoid and walks backwards. It then turns to the left and walks forward. If there is another such object in its way, it continues the same process until its path is clear and heads towards the light.

Spider Walker uses its two LED’s on the front of its head to signal whenever it’s turning right, left, or walking forwards. This works very similar to a turning signal. If Spider Walker comes across an overhanging object, it uses its height detection system to size up the object and decide if it can crawl under it or not. If it can crawl underneath it, Spider Walker slowly lowers itself to crawling height and proceeds to crawl. While crawling, Spider Walker sounds a steady beeping pattern to indicate that it is in crawl mode. It continues this until the rear IR sensor first sees the overhang and then decides that the overhang has been passed when it no long sees it. This is important, in that otherwise Spider Walker would not know how long the overhang was, and thus, how long to remain crawling.

Once passed the overhanging object, Spider Walker stops beeping, slowly stands up, and begins walking towards the light again. When it finally comes within proximity of the light, it orients itself with the light, flashes and beeps rapidly for about 30 seconds to indicate that it’s found the light, and then lowers itself into crawl mode. Once in crawl mode, it walks backwards away from the light, turns around, and drops off the first object. It then takes two more steps forward and drops off the second object and stops.

Experimental Layout and Results

I tested the height detection sensor by first standing the robot up at its approximate walking height. I then placed an object in front of the bottom sensor with the top sensor’s range of view unobstructed. Then the opposite was done. This provided me with reasonable data and an indication as to whether the sensor setup does its job. Tables 1 and 2 along with figures 16 and 17 illustrate the results of the testing.

Table 1. Height detection sensor output values as a function of obstructing obstacle's distance for **top detector obstruction**.

Distance (inches)	Top Detector Output	Bottom Detector Output
1	72	3
2	65	4
3	136	3
4	119	2
5	98	6
6	86	2
7	77	3
8	69	3

A more graphically pleasing result is depicted in figure 16 below. Note the sharp spike in the top sensor curve within the output threshold of concern (output values above 100).

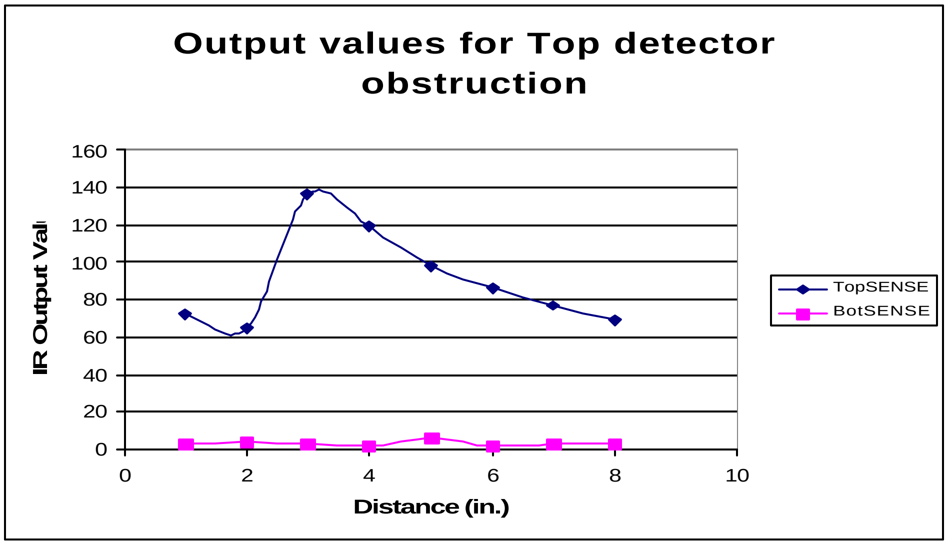

Figure 16. **Output values for Top detector obstruction**

Table 2. Height detection sensor output values as a function of obstructing obstacle's distance for **bottom detector obstruction.**

Distance(inches)	Top Detector Output	Bottom Detector Output
1	4	77
2	3	60
3	8	134
4	4	118
5	4	98
6	7	84
7	3	75
8	6	68

Figure 17 shows how the above behavior looks graphically. For this case, also note the sharp spike in the top sensor curve within the output threshold of concern. These last two figures are almost identical, with the exception of which curve goes with which sensor.

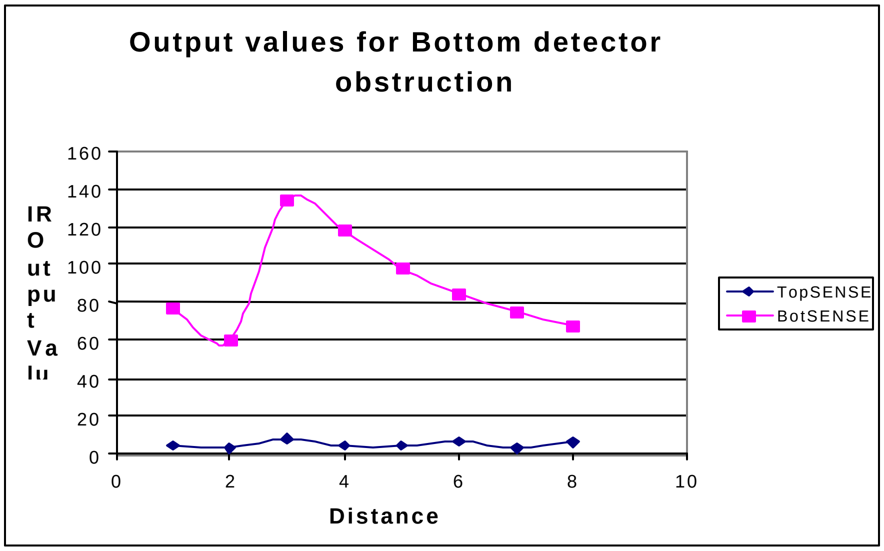

Figure 17. **Output values for bottom detector obstruction.**

Conclusion And Future Work

I am very happy to say that I more than exceeded my initial goals for this project. I did complete the robot and even had time to add extra gadgets and even put a nice paint job on it. Having a mechanical engineering background allowed me to do a complex mechanically designed robot in a relatively quick amount of time. It's the electrical aspect of the project that really took the most effort, particularly spending about a month learning how to interface my two boards using the SCI.

This was without question the most time consuming course I have ever taken in my life. And without question, it was by far the most rewarding. I don't think I learned this much material in one semester of any other class, including "THERMO II." This stuff was both theoretically enriching and unsurpassed as far has practical hands-on experience.

Developing this robot helped me learn how to conceptualize, design, build, test, and fix a product. If I did any future work on Spider Walker, it would probably be replacing the two AA 6 packs with one long lasting rechargeable battery cell. This would allow the robot to walk around for enough time on its own for me to really see how it interacts with a large, cluttered environment.

Documentation

Anita M. Flynn, Bruce A. Seiger, and Joseph L. Jones. *Mobile Robots: Inspiration to Implementation;* A.K. Peters, 1999.

David Novick, Keith L. Doty. *ROBOBUG Assembly Manual*; Mekatronix 1999.

Keith L. Doty. *TJPro Assembly Manual*; Mekatronix 1999.

Keith L. Doty. *TJ Assembly Manual*; Mekatronix 1999.

Acknowledgements

-The IMDL TA's, Aamir Qauyumi, Ariel Rodriquez, Tae Choi, for all their great knowledge

- Dr. Keith Doty	- Gave me extremely valuable advice throughout semester
- Dr. Eric Schwartz	- Assistant professor, great SCI tips
- Dr. A. Arroyo	- IMDL course professor/lecture, very helpful advice
- Scott Nortman	- Helped tremendously with windows XP problems

Sources for Parts

Mekatronix	-Both micro-controller boards, communications board, and computer cable
Radio Shack	-3.0 – 16VDC, 4.1kHz, 7 mA Piezo Mini Buzzer
IMDL	-LED's, resistors, wiring, 1/8" model airplane plywood

Total robot cost:	over $600 excluding

Appendicies

This is the code I wrote for the MSCC11 "servo board":

```
/*************************************************************************
 * Title            Spider_Walker_MSCC11_Servo_Board_Code.c                    *
 * Programmer   Garew E. Walker,                                                *
 *                  Initial ideas from Marc Poe's Hannibal Code                *
 * Date             April 11, 2002                                              *
 * Version        1                                                             *
 * Description: Code running on the MSCC11 board.  This code is my             *
 *                serial interface which receives ASCII codes from the         *
 *                SCI port to generate PWM signals for controlling up to       *
 *               16 servos.  It uses the SuperServo software I purchase        *
 *                from Mekatronix.                                             *
 *************************************************************************/

/****************************  BEGIN INCLUDES  **************************/
#include <bgbase.h>
#include <stdio.h>
#include <servobg.h>
#include <vectors.h>
#include <hc11.h>
#include <mil.h>
/**************************** END INCLUDES  ***************************/

/*************************  BEGIN MAIN   ********************************/
void main(void)
 {

   int actualpw,sn,pw;
         init_serial();
         init_servos();

   while(1)
    {
     sn = getchar();

     if (sn==0)
       {
        sn = getchar();                  // begin handshaking
        putchar(sn);
        pw=getchar();
        putchar(pw);
        actualpw=55*pw;        // convert to correct pwm value
        servo(sn,actualpw);              // comand the servo
        putchar(0x21);       // end transfer of data

       }
    }
 }
```

```
/****************************** END OF MAIN **************************/

/****************************  BEGIN FUNCTION DEFINITIONS **************/

void init_serial(void)
/*********************************************************************
 * Function description:                                              *
 *   Initializes the SCI port on the 68HC11 to operate at 9600 baud.  *
 *   This function must be called at the beginning of your program if *
 *   you wish to use any of the functions in this library.            *

 *********************************************************************/
{
  CLEAR_BIT(SPCR,0x20);
  BAUD = 0xb0;  /* 0xbO is 9600 0x35 is 300 baud */
  SCCR2 = 0x0C;
}
/************************End init_serial *****************************/


/************************* END FUNCTION DEFINITIONS *******************/
```

Below is my code which was loaded unto the TJPro11 "brain" board.

```
/*******************************************************************************
*
*
* Title          SPIDER_WALKER_ROBOT_BRAIN_BOARD_CODE.c
* Programmers    Garew E.-L. Walker
* Date           April 11, 2002
* Version        1
*
********************************************************************************
*
*
* Description:  This is the code running on the TJPro microcontroller board.
*               This board has a Motorola MC68HC11A1FN microprocessor for a
*               brain.  This code supplies all the robot'sintelligence by
*               monitoring all the sensors and communicating the information
*               with the single-chip board controlling all the servos.
*
*Serial Protocol of this program:
*  baud=9600, 8 data bits, 1 stop bit, no parity, no flow control.
*
* Example:
*   Configure Hyperterm on Win95:
*   Direct to COM1, 9600 baud, 8 bits, no parity, 1 stop bit,
*   no flow control.
*
*******************************************************************************/
```

```
/********************************* INCLUDES ***********************************/
#include <tjpbase.h>
#include <stdio.h>
#include <calbug.h>
/***************************** END OF INCLUDES *****************************/


/********************************* CONSTANTS *********************************/
#define TOP_IR       analog(2)   // PE2
#define BOTTOM_IR    analog(3)   // PE3
#define BACK_IR      analog(1)   // PE1
#define LEFT_CDS     analog(5)   // PE5
#define MIDDLE_CDS   analog(4)   // PE4
#define RIGHT_CDS    analog(6)   // PE6

#define stand_up_time 800   // 200 centiseconds = 2 seconds
#define turn_pause_time 800
#define crawl_start_time 1400
#define crawl_end_time 1400
#define sh_delay 190   // 200
#define leg_delay 390  // 400
#define quick2  60
#define quick1  70
/****************************** END OF CONSTANTS *****************************/


/********************************* PROTOTYPES  ******************************/
void move_servo(int sn, int pwm);
void setup_dig_port(void);
void stand_up_robot(void);
void Robot_raise_to_stand(void);
void crawl_stance_robot(void);
void Robot_lower_to_crawl(void);
void crawl(void);
void raise_leg(int leg, int pwm);
void lower_leg(int leg, int pwm);
void leg_forward(int leg, int pwm);
void leg_back(int leg, int pwm);
void tripod_gait_upright(int pwm_1up, int pwm_2up, int pwm_3up , int pwm_4up, int pwm_5up, int
pwm_6up, int wait1, int wait2);
void tripod_gait_crawl(int pwm_1crw, int pwm_2crw, int pwm_3crw,int pwm_4crw, int pwm_5crw, int
pwm_6crw, int wait1, int wait2);
void tripod_walk_backwards(int pwm_1up, int pwm_2up, int pwm_3up , int pwm_4up, int pwm_5up, int
pwm_6up, int wait1, int wait2);
void turn_left(int pwm_1, int pwm_2, int pwm_3, int pwm_4, int pwm_5, int pwm_6, int wait1, int wait2);
void turn_right(int pwm_1, int pwm_2, int pwm_3, int pwm_4, int pwm_5, int pwm_6, int wait1, int
wait2);
void obstacle_avoid(int backupsteps, int number_turns);
void momentary_straight_walk(int steps);
void momentary_straight_crawl(int steps);
void turn_left_number_times(int left_turns);
void turn_right_number_times(int right_turns);
void turn_left_crawl_number_times(int left_turns);
void turn_right_crawl_number_times(int right_turns);
void first_object_drop(void);
void second_object_drop(void);
void Right_Flash_On(void);
void Right_Flash_Off(void);
void Left_Flash_On(void);
void Left_Flash_Off(void);
void Speak_On(void);
void Speak_Off(void);
/*****************************  END OF PROTOTYPES  ***************************/



/**********************************  GLOBALS  *******************************/
 int backup_time = 3, backup_count;   //  CHANGE BACKUP_TIME TO ABOUT 5
 int turn_time = 3, turn_count;     //  CHANGE TURN_TIME TO ABOUT 6
 int walk_count;
 int crawl_ending_count;
 int end_main_loop_value;
 int light_found_value;
 int leg1_pwm, leg2_pwm, leg3_pwm, leg4_pwm, leg5_pwm, leg6_pwm;
 int pwm_change, smallest_pwm_delta = 14;
 int found = 1;
 int stop_looping = 1;
 int keep_looping = 0;
 int drop_pwm_delta;
 int speak_count;

/*******************************  END OF GLOBALS  ****************************/



/*********************************  BEGIN MAIN  *****************************/
void main(void)
  {


             init_analog();
             init_clocktjp();
             init_serial();
            setup_dig_port();

             IRE_ON;         /*turn on IR emitters */


   stand_up_robot();   // Stand up the robot

   wait(3000);   // wait for a little
```

```
            while(end_main_loop_value == keep_looping)   /* Main while loop */

                        {

                         while ((LEFT_CDS > 15) && (MIDDLE_CDS > 12) && (RIGHT_CDS > 15) &&
(TOP_IR < 100) && (BOTTOM_IR < 97))   // while #2

                          {
                            if ((LEFT_CDS < MIDDLE_CDS) && (LEFT_CDS < RIGHT_CDS))   // if #1
                             {
                               if ((LEFT_CDS < MIDDLE_CDS - 65) && (LEFT_CDS < RIGHT_CDS  - 65))
                                {
                                  wait(turn_pause_time);
                                  stand_up_robot();
                                  wait(turn_pause_time);

                                  turn_left_number_times(2);

                                  wait(turn_pause_time);
                                  stand_up_robot();
                                  wait(turn_pause_time);

                                  momentary_straight_walk(2);
                                }
                               else if ((LEFT_CDS < MIDDLE_CDS) && (LEFT_CDS < RIGHT_CDS))
                                {
                                  wait(turn_pause_time);
                                  stand_up_robot();
                  wait(turn_pause_time);

                  turn_left_number_times(2);

                  wait(turn_pause_time);
                  stand_up_robot();
                  wait(turn_pause_time);

                                  momentary_straight_walk(2);
                                }
                             }      // end if #1


                            else if ((MIDDLE_CDS < LEFT_CDS) && (MIDDLE_CDS < RIGHT_CDS))  //
else if #1
                             {
                               if ((MIDDLE_CDS < MIDDLE_CDS - 100) && (MIDDLE_CDS <
RIGHT_CDS - 100))
                                {
                                  momentary_straight_walk(3);
                                }
                               else if ((MIDDLE_CDS < LEFT_CDS) && (MIDDLE_CDS < RIGHT_CDS))
                                {
                                  momentary_straight_walk(2);
                                }
                             }       // end else if #1
```

```
                    else if ((RIGHT_CDS < LEFT_CDS) && (RIGHT_CDS < MIDDLE_CDS))   //
else if #2
                     {
                       if ((RIGHT_CDS < LEFT_CDS - 65) && (RIGHT_CDS < MIDDLE_CDS - 65))
                       {

                         wait(turn_pause_time);
            stand_up_robot();
            wait(turn_pause_time);

            turn_right_number_times(2);  // MAKE THIS 3

                         wait(turn_pause_time);
            stand_up_robot();
            wait(turn_pause_time);

                         momentary_straight_walk(2);
                       }
                      else if ((RIGHT_CDS < LEFT_CDS) && (RIGHT_CDS < MIDDLE_CDS))
                       {
                         wait(turn_pause_time);
            stand_up_robot();
            wait(turn_pause_time);

                        turn_right_number_times(2);  // Try only 1 time as well

                         wait(turn_pause_time);
            stand_up_robot();
            wait(turn_pause_time);

                         momentary_straight_walk(2);
                       }
                    }      // end else if #2


                 }  // end while #2



                if ((LEFT_CDS < 15) || (MIDDLE_CDS < 11) || (RIGHT_CDS < 15))  // start main if
                 {

                  if (MIDDLE_CDS < 11)
                    {
                     light_found_value = found;
                     end_main_loop_value = stop_looping;
                    }

                  else
                    {
```

```
                    while (MIDDLE_CDS > 11)
                      {
                       wait(stand_up_time);
                       stand_up_robot();
                       wait(stand_up_time);

                       if (LEFT_CDS < 15)
                         {
                          turn_left_number_times(1);
                         }

                       if (MIDDLE_CDS < 11)
                         {
                          light_found_value = found;
                          end_main_loop_value = stop_looping;
                         }

                       if (RIGHT_CDS < 15)
                         {
                          turn_right_number_times(1);
                         }

                      } // end while
                    }  // end else
                  }  // end main if



                else if ((TOP_IR > 95) || (BOTTOM_IR > 97))
                  {
                   if ((TOP_IR > 95) && (BOTTOM_IR < 100))
                     {
                      crawl();
                     }
                   else if (BOTTOM_IR > 97)
                     {
                      obstacle_avoid(3,3);  // can change to (4,3) or whatever works
                     }
                  }


              }   // end of main while loop


       if (light_found_value == found)
         {

          wait(1000);

          for (speak_count = 0; speak_count < 100; speak_count++)
            {
             Speak_On();  Right_Flash_On();  Left_Flash_On(); wait(60);
             Speak_Off();  Right_Flash_Off();  Left_Flash_Off(); wait(60);
            }
```

```
        wait(crawl_start_time);
        Robot_lower_to_crawl();
        wait(crawl_start_time);


              for (backup_count=0; backup_count < 2; backup_count++)
         {
          tripod_walk_backwards(70, 69, 60, 43, 59, 40, leg_delay, sh_delay);
         }

        crawl_stance_robot();

        turn_left_crawl_number_times(8);

        wait(crawl_start_time);
        crawl_stance_robot();
        wait(crawl_start_time);


        first_object_drop();

        momentary_straight_crawl(2);

        second_object_drop();

        momentary_straight_crawl(3);


             }
      }  // ****************** END OF MAIN PROGRAM *****************************



/*********************** BEGINING FUNCTION DEFINITIONS ********************/

void move_servo(int sn, int pwm)
 {
   int  i;
   char sn_receive, pwm_receive, end_transfer;

   put_char(0);
   put_char(sn);
   sn_receive = get_char();
   put_char(pwm);
   pwm_receive = get_char();
   end_transfer = get_char();
     while(end_transfer != 0x21)
      {}
 }


void setup_dig_port(void)
 {
   SET_BIT(DDRD, 0x08);  // Left LED bit
   SET_BIT(DDRD, 0x10); //  Right LED bit
   SET_BIT(DDRD, 0x04);  // Buzzer bit

   SET_BIT(PORTD, 0x08);
   SET_BIT(PORTD, 0x10);
   CLEAR_BIT(PORTD, 0x04);
 }


void stand_up_robot(void)
 {
   move_servo(4,52);
   move_servo(5,72);
   move_servo(6,54);
   move_servo(7,63);
   move_servo(8,44);
   move_servo(9,49);
   move_servo(3,58);  //  This servo pin is broken move_servo(10,62);
   move_servo(11,64);
   move_servo(12,74);
   move_servo(13,59);
   move_servo(14,55);
   move_servo(15,44);
 }


void crawl_stance_robot(void)
 {
   move_servo(4,70);
   move_servo(5,72);
   move_servo(6,69);
   move_servo(7,63);
   move_servo(8,60);
   move_servo(9,49);
   move_servo(3,43);//  Replaces move_servo(10,58) because pin malfunctioned
   move_servo(11,64);
   move_servo(12,59);
   move_servo(13,59);
   move_servo(14,40);
   move_servo(15,44);
 }


void Robot_lower_to_crawl(void)
 {
   stand_up_robot();

   leg1_pwm = 52;
   leg2_pwm = 54;
   leg3_pwm = 44;
   leg4_pwm = 58;
   leg5_pwm = 74;
   leg6_pwm = 55;
   for (pwm_change = 0; pwm_change < smallest_pwm_delta; pwm_change++)
     {
       raise_leg(1, leg1_pwm + 1);  leg1_pwm =leg1_pwm + 1;  wait(150);
       raise_leg(2, leg2_pwm + 1);  leg2_pwm =leg2_pwm + 1;  wait(150);
       raise_leg(3, leg3_pwm + 1);  leg3_pwm =leg3_pwm + 1;  wait(150);
       raise_leg(4, leg4_pwm  - 1);  leg4_pwm =leg4_pwm  - 1;  wait(150);
       raise_leg(5, leg5_pwm  - 1);  leg5_pwm =leg5_pwm  - 1;  wait(150);
       raise_leg(6, leg6_pwm  - 1);  leg6_pwm =leg6_pwm  - 1;  wait(150);
     }

   wait(200);
   crawl_stance_robot();
 }



void Robot_raise_to_stand(void)
{
 crawl_stance_robot();

 leg1_pwm = 70;
 leg2_pwm = 69;
 leg3_pwm = 60;
 leg4_pwm = 43;
 leg5_pwm = 59;
 leg6_pwm = 40;


 for (pwm_change = 0; pwm_change < smallest_pwm_delta; pwm_change++)
   {
     lower_leg(1, leg1_pwm - 1);  leg1_pwm =leg1_pwm - 1;  wait(150);
     lower_leg(2, leg2_pwm - 1);  leg2_pwm =leg2_pwm - 1;  wait(150);
     lower_leg(3, leg3_pwm - 1);  leg3_pwm =leg3_pwm - 1;  wait(150);
     lower_leg(4, leg4_pwm + 1);  leg4_pwm =leg4_pwm + 1;  wait(150);
     lower_leg(5, leg5_pwm + 1);  leg5_pwm =leg5_pwm + 1;  wait(150);
     lower_leg(6, leg6_pwm + 1);  leg6_pwm =leg6_pwm + 1;  wait(150);
   }

 wait(200);
 stand_up_robot();
}


void raise_leg(int leg, int pwm)
 {
   switch(leg)
             {
               case 1: move_servo(2*leg+2,pwm); break ;
                       case 2: move_servo(2*leg+2,pwm); break ;
                       case 3: move_servo(2*leg+2,pwm); break ;
               case 4: move_servo(leg - 1,pwm); break ;
                       case 5: move_servo(2*leg+2,pwm); break ;
               case 6: move_servo(2*leg+2,pwm); break ;
               default: break;
    }
  }



void lower_leg(int leg, int pwm)
  {
   switch(leg)
              {
               case 1: move_servo(2*leg+2,pwm); break ;
               case 2: move_servo(2*leg+2,pwm); break ;
               case 3: move_servo(2*leg+2,pwm); break ;
               case 4: move_servo(leg - 1,pwm); break ;
               case 5: move_servo(2*leg+2,pwm); break ;
               case 6: move_servo(2*leg+2,pwm); break ;
               default: break;
     }
  }



void leg_forward(int leg, int pwm)
  {
   switch(leg)
              {
               case 1: leg=2*leg+3; move_servo(leg, pwm); break ;
                        case 2: leg=2*leg+3; move_servo(leg, pwm); break ;
                        case 3: leg=2*leg+3; move_servo(leg, pwm); break ;
               case 4: leg=2*leg+3; move_servo(leg, pwm); break ;
                        case 5: leg=2*leg+3; move_servo(leg, pwm); break ;
               case 6: leg=2*leg+3; move_servo(leg, pwm); break ;
               default: break;
              }
  }


void leg_back(int leg, int pwm)
  {
   switch(leg)
              {
               case 1: leg=2*leg+3; move_servo(leg, pwm); break ;
                        case 2: leg=2*leg+3; move_servo(leg, pwm); break ;
                        case 3: leg=2*leg+3; move_servo(leg, pwm); break ;
               case 4: leg=2*leg+3; mo ve_servo(leg, pwm); break ;
                        case 5: leg=2*leg+3; move_servo(leg, pwm); break ;
               case 6: leg=2*leg+3; move_servo(leg, pwm); break ;
               default: break;
              }
  }


void crawl(void)
  {
    wait(crawl_start_time);
    Robot_lower_to_crawl();
    wait(crawl_start_time);

    while ((BACK_IR < 75) && (BOTTOM_IR < 100) && (LEFT_CDS > 15) && (MIDDLE_CDS > 12)
&& (RIGHT_CDS > 15))
      {
        tripod_gait_crawl(70, 69, 60, 43, 59, 40, leg_delay, sh_delay);
      }

    while ((BACK_IR > 75) && (BOTTOM_IR < 100) && (LEFT_CDS > 15) && (MIDDLE_CDS > 12)
&& (RIGHT_CDS > 15))
      {
        tripod_gait_crawl(70, 69, 60, 43, 59, 40, leg_delay, sh_delay);
      }

    for (crawl_ending_count = 0; crawl_ending_count < 1; crawl_ending_count++)
      {
        if (BOTTOM_IR < 100)
          {
            tripod_gait_crawl(70, 69, 60, 43, 59, 40, leg_delay, sh_delay);  // Averaged values
          }
      }

    if ((BACK_IR < 50) && (BOTTOM_IR < 100))
      {
        wait(crawl_end_time);
                Robot_raise_to_stand();
                wait(crawl_end_time);
      }
  }


void momentary_straight_walk(int steps)
  {
    for (walk_count = 0; walk_count < steps; walk_count++)
      {
        if ((BOTTOM_IR < 97) && (TOP_IR < 95) && (LEFT_CDS > 15) && (MIDDLE_CDS > 12) &&
(RIGHT_CDS > 15))
          {
            tripod_gait_upright(52,54,44,58,74,55, leg_delay, sh_delay);
          }
      }

    wait(100);   // Delay robot a little before it straightens out
    stand_up_robot();
    wait(100);   // and delay it a little after is straightens out

  }


void momentary_straight_crawl(int steps)
  {
    for (walk_count = 0; walk_count < steps; walk_count++)
    {
      if ((BOTTOM_IR < 97) && (TOP_IR < 95) && (LEFT_CDS > 15) && (MIDDLE_CDS > 12) &&
(RIGHT_CDS > 15))
         {
           tripod_gait_crawl(70, 69, 60, 43, 59, 40, leg_delay, sh_delay);
         }
     }

    wait(100);   // Delay robot a little before it straightens out
    crawl_stance_robot();
    wait(100);   // and delay it a little after is straightens out

  }


void obstacle_avoid(int backupsteps, int number_turns)
  {
    backup_time = backupsteps;
    turn_time = number_turns;

    wait(stand_up_time);
    stand_up_robot();
    wait(stand_up_time);


    for (speak_count = 0; speak_count < 3; speak_count++)
     {

       Speak_On();  wait(800);
       Speak_Off(); wait(500);
       Speak_On();  wait(100);
       Speak_Off(); wait(300);
       Speak_On();  wait(100);
       Speak_Off(); wait(300);
       Speak_On();  wait(100);
       Speak_Off(); wait(300);
       Speak_On();  wait(40);
       Speak_Off(); wait(100);
       Speak_On();  wait(40);
       Speak_Off(); wait(100);
       Speak_On();  wait(40);
       Speak_Off(); wait(100);
     }


    for (backup_count=0; backup_count < backup_time; backup_count++)
     {
       tripod_walk_backwards(52,54,44,58,74,55, leg_delay, sh_delay);
     }

    wait(stand_up_time);
    stand_up_robot();
    wait(stand_up_time);
```

```
   for (turn_count = 0; turn_count < turn_time; turn_count++)
      {
        turn_left(52,54,44,58,74,55, leg_delay, sh_delay);
      }

    momentary_straight_walk(5);

  }


void tripod_gait_upright(int pwm_1up, int pwm_2up, int pwm_3up , int pwm_4up, int pwm_5up, int
pwm_6up, int wait1, int wait2)
  {


    raise_leg(1,83);       raise_leg(3,73);       raise_leg(5,47);       wait(wait1);
    leg_forward(1,72);     leg_forward(3,53);     leg_forward(5,52);     wait(wait2);
    lower_leg(1, pwm_1up); lower_leg(3, pwm_3up); lower_leg(5, pwm_5up); wait(wait1);
    leg_back(1,68);        leg_back(3,49);        leg_back(5,59);        wait(wait2);

    raise_leg(2,82);       raise_leg(4,31);       raise_leg(6,29);       wait(wait1);
    leg_forward(2,69);     leg_forward(4,59);     leg_forward(6,41);     wait(wait2);
    lower_leg(2, pwm_2up); lower_leg(4, pwm_4up); lower_leg(6, pwm_6up); wait(wait1);
    leg_back(2,63);        leg_back(4,64);        leg_back(6,51);        wait(wait2);

  }


void tripod_gait_crawl(int pwm_1crw, int pwm_2crw, int pwm_3crw,int pwm_4crw, int pwm_5crw, int
pwm_6crw, int wait1, int wait2)
  {

    raise_leg(1,83);        raise_leg(3,73);        raise_leg(5,47);        wait(wait1);   Speak_On();
    leg_forward(1,72);      leg_forward(3,53);      leg_forward(5,52);      wait(wait2);   Speak_Off();
    lower_leg(1, pwm_1crw); lower_leg(3, pwm_3crw); lower_leg(5, pwm_5crw); wait(wait1);
    leg_back(1,68);         leg_back(3,49);         leg_back(5,59);         wait(wait2);


    raise_leg(2,82);        raise_leg(4,31);        raise_leg(6,29);        wait(wait1);   Speak_On();
    leg_forward(2,69);      leg_forward(4,59);      leg_forward(6,41);      wait(wait2);   Speak_Off();
    lower_leg(2, pwm_2crw); lower_leg(4, pwm_4crw); lower_leg(6, pwm_6crw); wait(wait1);
    leg_back(2,63);         leg_back(4,64);         leg_back(6,51);         wait(wait2);

  }




void tripod_walk_backwards(int pwm_1up, int pwm_2up, int pwm_3up , int pwm_4up, int pwm_5up, int
pwm_6up, int wait1, int wait2)
```

```
{

   raise_leg(1,83);      raise_leg(3,73);      raise_leg(5,47);      wait(wait1);
   leg_back(1,58);       leg_back(3,49);       leg_back(5,72);       wait(wait2);
   lower_leg(1, pwm_1up); lower_leg(3, pwm_3up); lower_leg(5, pwm_5up); wait(wait1);
   leg_forward(1,72);    leg_forward(3,53);    leg_forward(5,59);    wait(wait2);

   raise_leg(2,82);      raise_leg(4,31);      raise_leg(6,29);      wait(wait1);
   leg_back(2,46);       leg_back(4,64);       leg_back(6,64);       wait(wait2);
   lower_leg(2, pwm_2up); lower_leg(4, pwm_4up); lower_leg(6, pwm_6up); wait(wait2);
   leg_forward(2,63);    leg_forward(4,59);    leg_forward(6,44);    wait(wait1);

  }



void turn_left(int pwm_1, int pwm_2, int pwm_3, int pwm_4, int pwm_5, int pwm_6, int wait1, int wait2)
  {
   raise_leg(2,82);    raise_leg(5,47);    wait(wait1);  Left_Flash_On();
   leg_forward(2,69);  leg_back(5,70);     wait(wait2);  Left_Flash_Off();
   lower_leg(2, pwm_2); lower_leg(5, pwm_5); wait(wait1);  Left_Flash_On();
   leg_back(2,51);              leg_forward(5,52);   wait(wait2);  Left_Flash_Off();

   raise_leg(4,31);    raise_leg(3,73);                Left_Flash_On();
   leg_forward(3,70);  leg_back(4,64);                   Left_Flash_Off();
   lower_leg(4, pwm_4); lower_leg(3, pwm_3); wait(wait1);  Left_Flash_On();
   leg_back(3,49);              leg_forward(4,42);   wait(wait2);  Left_Flash_Off();

   raise_leg(1,83);    raise_leg(6,29);                Left_Flash_On();
   leg_forward(1,72);  leg_back(6,64);                   Left_Flash_Off();
   lower_leg(1, pwm_1); lower_leg(6, pwm_6); wait(wait1);  Left_Flash_On();
   leg_back(1,58);              leg_forward(6,44);   wait(wait2);  Left_Flash_Off();
  }



void turn_right(int pwm_1, int pwm_2, int pwm_3, int pwm_4, int pwm_5, int pwm_6, int wait1, int wait2)
  {
   raise_leg(2,82);    raise_leg(5,47);    wait(wait1);  Right_Flash_On();
   leg_forward(5,52);  leg_back(2,51);     wait(wait2);  Right_Flash_Off();
   lower_leg(2, pwm_2); lower_leg(5, pwm_5); wait(wait1);  Right_Flash_On();
   leg_back(5,70);              leg_forward(2,69);   wait(wait2);  Right_Flash_Off();

   raise_leg(4,31);    raise_leg(3,73);                Right_Flash_On();
   leg_forward(4,42);  leg_back(3,49);                   Right_Flash_Off();
   lower_leg(4, pwm_4); lower_leg(3, pwm_3); wait(wait1);  Right_Flash_On();
   leg_back(4,64);              leg_forward(3,70);   wait(wait2);  Right_Flash_Off();

   raise_leg(1,83);    raise_leg(6,29);                Right_Flash_On();
   leg_forward(6,44);  leg_back(1,58);                   Right_Flash_Off();
   lower_leg(1, pwm_1); lower_leg(6, pwm_6); wait(wait1);  Right_Flash_On();
   leg_back(6,64);              leg_forward(1,72);   wait(wait2);  Right_Flash_Off();
  }
```

```
void turn_left_number_times(int left_turns)
  {
    for (turn_count = 0; turn_count < left_turns; turn_count++)
      {
        if ((BOTTOM_IR < 97) && (TOP_IR < 95))
          {
            turn_left(52,54,44,58,74,55, leg_delay, sh_delay);
          }
      }
  }


void turn_right_number_times(int right_turns)
  {
    for (turn_count = 0; turn_count < right_turns; turn_count++)
      {
        if ((BOTTOM_IR < 97) && (TOP_IR < 95))
          {
            turn_right(52,54,44,58,74,55, leg_delay, sh_delay);
          }
      }
}


void turn_left_crawl_number_times(int left_turns)
  {
    for (turn_count = 0; turn_count < left_turns; turn_count++)
      {
        if ((BOTTOM_IR < 97) && (TOP_IR < 95))
          {
            turn_left(70, 69, 60, 43, 59, 40, leg_delay, sh_delay);
          }
      }
  }


void turn_right_crawl_number_times(int right_turns)
  {
    for (turn_count = 0; turn_count < right_turns; turn_count++)
      {
        if ((BOTTOM_IR < 97) && (TOP_IR < 95))
          {
            turn_right(70, 69, 60, 43, 59, 40, leg_delay, sh_delay);
          }
      }
  }


void first_object_drop(void)
  {
    int dropper_current_pwm = 32;
    drop_pwm_delta = 27;
  for (pwm_change = 0; pwm_change < drop_pwm_delta ; pwm_change++)
     {
        move_servo(2, dropper_current_pwm + 1);
       dropper_current_pwm = dropper_current_pwm + 1;  wait(150);
     }
  }


void second_object_drop(void)
  {
    int dropper_current_pwm = 59;
    drop_pwm_delta = 20;

    for (pwm_change = 0; pwm_change < drop_pwm_delta ; pwm_change++)
      {
        move_servo(2, dropper_current_pwm + 1);
       dropper_current_pwm = dropper_current_pwm + 1;  wait(150);
      }
  }


void Right_Flash_On(void)
  {
   CLEAR_BIT(PORTD, 0x08);
  }


void Right_Flash_Off(void)
  {
   SET_BIT(PORTD, 0x08);
  }


void Left_Flash_On(void)
  {
    CLEAR_BIT(PORTD, 0x10);
  }

void Left_Flash_Off(void)
  {
    SET_BIT(PORTD, 0x10);
  }

void Speak_On(void)
  {
    SET_BIT(PORTD, 0x04);
  }

  void Speak_Off(void)
  {
    CLEAR_BIT(PORTD, 0x04);
  }

/**************** END FUNCTION DEFINITIONS  ********************/
```