Robot: Julius

Final Report

Joseph P. Osentowski

EEL 5666

Intelligent Machines Design Laboratory

Dr. A. Arroyo

Dr. E. Schwartz

Table of Contents

Abstract:

Orange growers rely heavily on paid workers to collect fruit from the trees in the grove. Fruit that has fallen to the ground is not viable for resale in a commercial enterprise. Robot:Julius is designed to collect this fruit and carry it away for disposal. This eliminates the need to pay someone to collect fruit that will only be disposed of anyway. Robot:Julius is designed to make use of a camera to identify oranges on the ground which are collected through an integrated collection system. Once enough oranges are collected, the robot will travel to a predetermined location to deposit its payload so that the oranges can be disposed of.

Robot:Julius consists of a basic four-wheeled platform, featuring a front-wheel, differential drive system. Obstacle avoidance is performed through the use of sonar and contact switches. Oranges are located using a webcam linked through a laptop computer mounted to the robot platform. The laptop uses OpenCV software to process images obtained from the camera to track the color orange. Once an orange is detected, the robot is designed to simply drive over it, picking it up with an integrated collection system.

Executive Summary:

Robot:Julius makes use of three sonar sensors (two front-mounted, one rear-mounted) for basic obstacle avoidance. Should the sonar fail to indicate an obstacle, the robot is equipped with two contact sensors on each side to indicate a collision. Should one of these be triggered, the robot will drive the front wheels in opposite directions, forcing the robot to back away from the object while simultaneously turning away from the side where the collision occurred. Should either the two front-mounted sonars, or the contact switched on both sides be triggered at the same time, the robot will sound a buzzer to indicate that it is going to reverse direction entirely. After backing away, Robot:Julius will turn away and renew the search for oranges to collect.

Oranges are identified by using a simple webcam connected to a laptop running OpenCV. The laptop is connected to the microcontroller board on the robot by a USB-to-Serial converter. The converter works properly, the microcontroller is capable of sending and receiving data through the converter, and the OpenCV program does reliably identify the color orange for "blob tracking." Unfortunately, the laptop does not perform the initial "handshake" necessary to initiate two-way communication. Without this connection, the robot cannot receive targeting data from the laptop. Therefore Robot:Julius is, for the time being, restricted to roaming behaviors only, collecting only those oranges (and other objects) that happen to be in its path.

Since the camera communication does not completely work, Robot:Julius' homing routine has been temporarily re-written. Once the pressure switch under the bottom plate of the basket is triggered, the robot will stop, sounding the buzzer three times in succession before opening the gate on the side of the basket to release the oranges that have been collected. Once the payload has been dropped, the robot resumes its roaming behavior.

There is a photoresistor mounted to the top of the robot platform to detect ambient light levels while the robot is in operation. Should the light level drop below a given threshold, an LED array will light, acting as a headlight. This will be used to reduce the effect of low light levels on the camera's ability to detect color, once the targeting issue is resolved.

Introduction:

Fruit typically falls from a tree for one of three reasons:

1. The fruit has reached maturity and is too heavy to remain on the branch
2. The tree is diseased, or otherwise weakened, and cannot support the weight of the fruit
3. The fruit has become infested with insect eggs, and has become too heavy due to the additional weight

Regardless of the reason, once the fruit has fallen to the ground, it is no longer desirable for the grower to do anything with the fruit but dispose of it. Robot: Julius is designed to collect this undesirable fruit, and deposit it into a receptacle, so that it can be disposed of or destroyed.

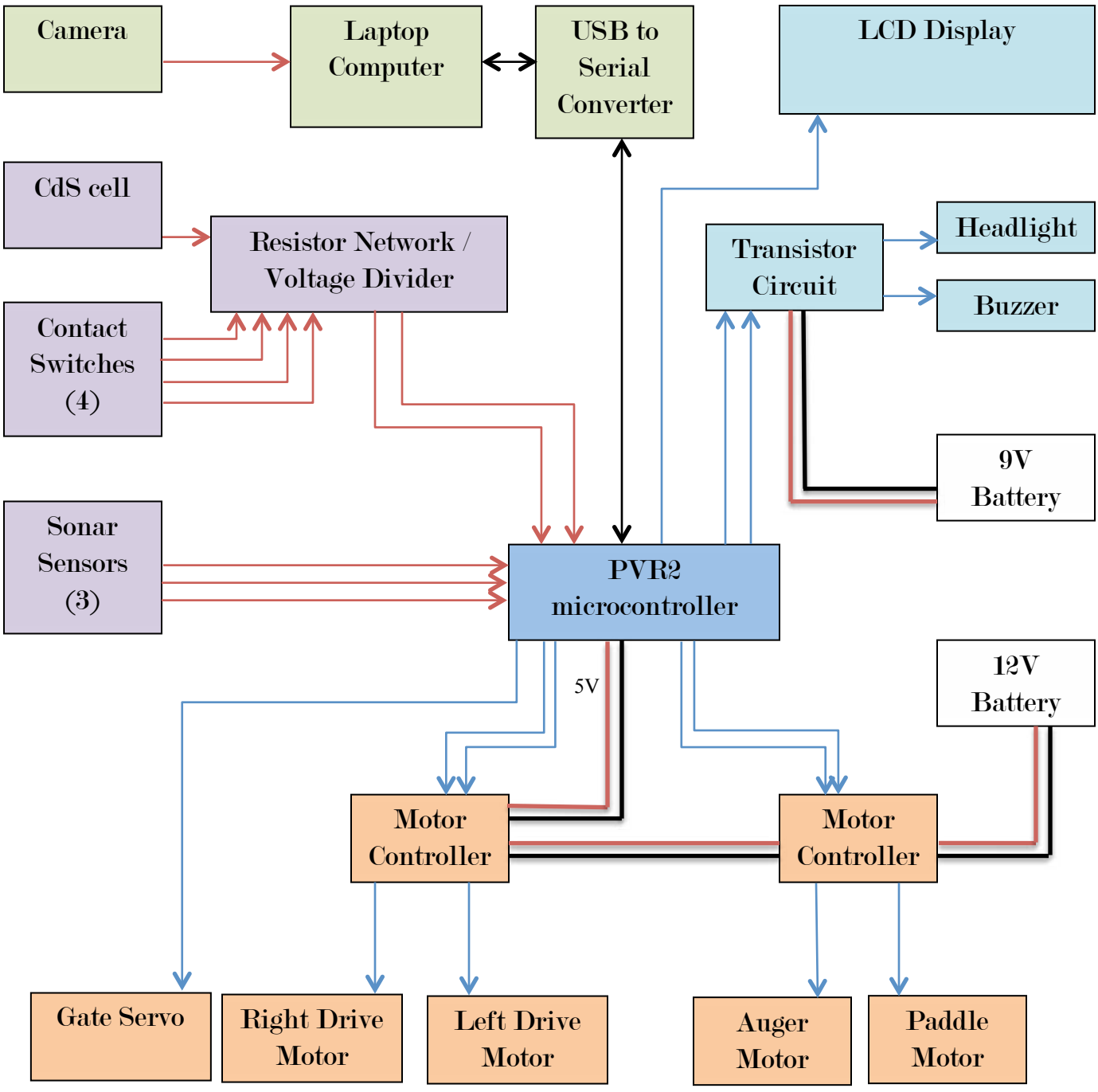
By regularly collecting this fallen fruit, the possibility of insect infestation is significantly reduced, as is the possibility of spreading plant diseases to neighboring trees.

Integrated System:

Robot:Julius utilizes the microcontroller board by Pridgen and Vermeer (PVR2). This controller makes use of the Atmel Xmega128A1 chip. This microcontroller performs all major processing tasks governing the various behaviors of the robot, with the exception of image processing for the camera, which is taken care of via a laptop computer that is physically mounted to the robot platform. Due to the higher voltage and current needed to power the motors, two motor controller boards are utilized by Robot:Julius. These controllers are from the Sabertooth line of motor controllers manufactured by Dimension Engineering. The control signals are generated from the PVR2 microcontroller board, via the PWM output. All of the

other electronic circuits are relatively simple. The resistor network for the contact sensors and the voltage divider circuit are both hand-soldered to a piece of perfboard, and mounted with the other electronics in the main enclosure. The transistor circuits used as switches for the buzzer and headlight are both hand-soldered as well, but have been placed in a separate enclosure along with a nine-Volt battery to power both devices.

A diagram of the integrated system follows:



It should be noted that in the diagram of the integrated system that:

1. The connection between the laptop computer and the USB to serial converter does not function at present.
2. The motor controller by Dimension Engineering does provide a regulated 5V output for the expressed purpose of powering onboard electronics. This is the connection used to power the PVR2 board.

Mobile Platform:

The platform must be rugged enough to function outdoors, on unfinished surfaces (i.e. grass). To make it resistant to adverse weather conditions, Robot: Julius is constructed primarily from sheet polycarbonate. Polycarbonate is strong, lightweight, and easily maintained.

The drive mechanism is a two-wheel, differential drive system. This gives the advantages of simple implementation, tight turning radius, and no additional steering linkages that could be damaged by obstacles such as tree roots. The motors for the drive system are geared at a ratio of 425:1. These turn very slowly, but draw little current, while providing plenty of torque. This allows the robot to move with a nearly constant speed, regardless of the terrain conditions. The slow turning wheels also have the added advantage of making targeting easier.

The motor for the paddle is a direct drive motor salvaged from a printer. It provides adequate power to keep the paddle turning rapidly, without stripping the connection between the motor shaft and the center shaft of the paddle. Ideally, this should be replaced with a gear motor. A gear motor would be less likely to bind, should an orange land in a "less than ideal" location, between the paddle and the chute.

The auger is a steel screw mounted to a wooden shaft. It is powered by a gear motor to keep it turning slowly with high torque. This keeps the auger turning even if the orange gets wedged between the edge of the screw and the side of the chute.

The basket is mounted to the underside of the platform. Once an orange reaches the end of the screw mechanism, it falls onto a raked platform. The orange can then roll freely to the end of basket where it is held by a small "gate" which can be opened by a servo. When enough pressure is exerted on the base of the basket (due to the weight of the oranges), the robot will stop, sound the buzzer to announce that it is changing routines, and open the gate to allow the oranges to roll out of the basket.

Robot: Julius is equipped with an LED headlight, so that it can more effectively operate in lower light conditions, such as when it is under a tree canopy. The headlights operate only when a low-light condition is determined, using a photo resistive cell to measure ambient lighting conditions.

Power for the motors and microcontroller is provided primarily by a single, twelve-volt, sealed lead-acid battery. This battery is the same type as those used for backup power supplies in security systems and other applications that may require emergency power. A single nine-Volt battery is used to power the buzzer and headlight, connected through a simple transistor circuit.

Actuation:

Beyond the differential drive system, there is a combine/conveyor system for collecting oranges, and a dumping system to drop the collected oranges into a suitable receptacle for disposal. The combine/conveyor system consists of a roller with a series of paddles attached (the combine), a collection chute, and a screw mechanism to transport the fruit to the basket (the auger). When the combine roller comes in contact with an orange, it will propel the orange further toward the center of the robot platform, into the chute. From the base of the chute, the auger transports the orange up the chute until it is deposited in the basket.

The auger design is being used instead of the originally designed conveyor belt system for three reasons. First, a conveyor belt requires a proper tension to work effectively, whereas the auger design simply needs to be designed with a proper pitch. Secondly, the orientation of the auger is not critical, as it is capable of gathering oranges from above, below, or the side with equal efficiency. Lastly, the conveyor belt design has the disadvantage of needing some way to ensure that the orange does not simply roll back down the belt. This is not a concern with the auger design, as the orange is held firmly, even if the auger stops moving.

Robot:Julius makes use of a basic webcam to track oranges on the ground. The data is sent from the camera to a laptop computer, and the images processed using OpenCV libraries. The laptop is supposed to send the tracking data to the PVR2 controller to adjust the movement of the robot. There are issues with the serial communication from the laptop, so the robot does not currently track oranges. This severely limits the overall scope of the project, as the primary goal of the project design is based on vision.

Sensors:

Robot: Julius has several sensors to respond to external stimuli:

Three ultrasonic sensors (sonar) are used for obstacle avoidance. Two of these are front facing, and once rear facing. The two front facing sensors are set at angles on the front corners of the robot, to avoid obstacles while driving. The rear mounted sensor is aimed straight backward, so that Robot: Julius does not run into anything (or anyone) while driving in reverse.

The Maxbotix LV-MaxSonar-EZ1 sonar module is being used for this purpose.

In the event that the sonar detectors fail, and Robot: Julius comes into contact with an obstacle, there is a series of contact sensors (bump switches) on the front and sides of the platform. These sensors are attached to two bumpers which extend from the front of the platform, around the sides of the drive wheels. Should any of these sensors be activated, Robot: Julius will reverse direction, while turning away from the offending obstacle for a predetermined time, and then resume whatever behavior it was engaged in before the switch was activated. All of the bump switches are networked together through a hand-made circuit mounted with the main electronics on a piece of perfboard. This circuit is based on that referenced in Mobile Robots: Inspiration to Implementation.

There is a single photo resistor aimed upward, to detect ambient light levels. This determines whether or not to engage the headlamp mounted on the front of the platform. The headlamp is equipped to aid the camera in color detection should the light level be significantly reduced, as would be the case when Robot: Julius travels beneath the tree canopy. This will ensure that the camera is functioning at its best in changing lighting conditions. This is linked to the PVR microcontroller via a voltage divider mounted to the same perfboard as the resistor network for the bump switches.

The camera is a basic computer webcam (Logitech C500). This is used to identify oranges on the ground, in front of Robot: Julius. The camera is fixed mount, and works in conjunction with the photo resistor on top of the platform to maximize its efficiency, should the ambient lighting change rapidly (i.e., due to cloud cover, or the robot moving under a tree canopy). All image processing tasks are performed by a laptop computer mounted to the robot platform itself. OpenCV is being used to perform color detection.

Sensor Behaviors:

The sensor behaviors are grouped based on sensor type. The main program cycles through the sensor routines in the following order:

1. Bump Sensors
2. Sonar Sensors
3. Pressure Switch
4. Camera
5. Cds Cell

The order is determined by evaluating the severity of the stimulus. Obstacle avoidance behaviors take a higher priority than the actual collection of oranges or changes in lighting conditions. The output of the motors is determined based on sensor input. The camera routine is not working. Ideally, it would be used to identify an orange using the webcam, and then engage the collection system as the robot approaches its target.

Motor Behaviors:

Cruise: The robot drives in a slow arc. This is the default behavior unless an obstacle is encountered.

Left Turn / Right Turn: One wheel slows while the other remains engaged at full speed. This causes the robot to turn in a given direction. These are governed by the sonar sensors on the front of the robot.

Hard Left / Hard Right: One wheel not only slows, but reverses direction, while the other remains engaged at full speed. This causes the robot to turn and back away slightly from an obstacle. This is governed by the bump switches on the front and sides of the robot.

Reverse: As the name implies, both motors will stop, and then reverse direction to move the robot backward. In the event that this routine is triggered, the buzzer will sound to alert anyone behind the robot. There is a single rear-facing sonar mounted to prevent the robot from backing into anyone careless enough to remain behind it after being warned by the buzzer. The robot will likely revert to the Cruise behavior after this routine. Given that the Cruise behavior leads the robot to travel in an arc, the robot should be able to avoid any obstacle triggering the Reverse routine.

Dump: When the pressure switch in the bottom of the basket is activated, the drive motors stop. The buzzer sounds three times to indicate that the payload is about to be dropped, and then the gate open to release the oranges contained within. Ideally, this would be done at a designated location. Since the camera communication is not functional, locating a drop-off site is not possible at present.

Experimental Layout and Results:

In its initial design, the basket was to be mounted on top of the platform, and the entire thing tipped to one side to empty its contents. This was changed for three major reasons. First, because the oranges are dropped into the basket from above, the transport system would be required to carry them much further. This leads to the second issue of the overall height of the robot. Placing the basket on top of the platform, and then having another mechanism reach above the rim of the basket would make the robot unnecessarily tall, and add additional weight due to the larger components. Orange trees have relatively low branches, so the overall height is a concern. Since there is plenty of unused space underneath the platform, it made more sense to place the basket there. Lastly, tipping the basket would require another motor capable of lifting the combined weight of the basket and its payload. It made more sense to simply empty the basket from the side, requiring little power, and letting the oranges simply roll out under the influence of gravity.

All sensors were tested as separate routines, sending the results to the LCD display. All of the sensors are interfaced through the analog-to-digital conversion (ADC) on the PVR2 board. Each sensor input is read as a value between 0 and 4095.

Once the values generated by each sensor (or combination of sensors in the case of the bump switches) were determined, a general obstacle avoidance routine was developed. This routine directly controlled the motors from the sensor input, which is less than ideal. This behavior was separated into a bump sensor routine, a sonar sensor routine, and a series of motor routines. This allows the programming code to run more smoothly, and to be modified more readily. Simply put, the code is easier to read, easier to write, and easier to change if it is done this way.

Next, the collection system was tested. Originally, the auger had a direct drive motor salvaged from a twelve-Volt electric drill. It was assumed that this would be capable of providing sufficient torque to keep the auger moving, even if an orange is stuck. This proved to be a mistake. Before demonstration, an orange bound in the auger caused it to overheat, ruining the motor contacts. This motor was replaced with a gear motor with a ratio of 120:1. The replacement motor proved far superior, as it turned slowly and steadily under varying load, without drawing excessive current.

The camera was the most daunting task. Getting OpenCV to recognize the camera was troublesome. The camera had to be changed more than once to find one that was compatible.

Apparently, different releases of OpenCV have different compatibility issues. The computer code is borrowed from Matt Morgan, who adapted it from code written by Cristobal Carnero Linan. This was easily adjusted to suit the needs of this project, requiring that the HSV settings be adjusted to track the color orange effectively. Getting the serial communication to work is not so easy. Again, the code from Matt Morgan's project was used, but it did not work. I searching through the Microsoft Developers' Network, other code was obtained. This code did not work either. The laptop can identify the communications port and adjust the settings, but not open it. Given that the serial communication would not work, the main goal of the project cannot be realized. All of the other components of the project do work, leaving this issue as the only remaining obstacle.

Finally, all of the working components were combined. The main program consists of few lines in itself, calling subroutines as needed. While this may not be the ideal way to write the program, it works for purposes of this project.

Conclusion:

Robot:Julius does not do everything originally envisioned. Getting the camera to work properly is what links many of the elements together. The camera works, and the microcontroller can communicate with the converter. The laptop recognizes the virtual port, but refuses to open it to send the camera data to the PVR board. Without it, there is no data to write a behavior routine to actively target oranges.

Even without the camera, the overall design does appear sound. The robot moves as expected, and the collection system does work as designed. Improvements could be made to the collection system by modifying the paddle in the front of the chute. A gear motor should be used. When the paddle strikes something sufficiently solid (like an orange), it slows considerably. This causes the current to spike, giving inconsistent performance. It has also been suggested that the paddle be reoriented (and possibly paired with a second paddle), so that the axis of the paddle is vertical. This would eliminate the need to notch the center of the paddle to allow it to pass around the center of the auger.

At its final demonstration, Robot:Julius failed to respond to a bumper being pressed. The LCD display indicated that the sonar had been triggered, and the robot was attempting to turn left. Why the second sonar did not trigger a reverse routine is unclear. What is clear is that the behavior needs to be modified to allow for other sensor inputs. The collection system failed to catch the first orange, but it did transport the second orange through the auger without any problems. The dump routine worked as written, though it is not the routine originally intended. Without the camera, an alternate routine was written merely to demonstrate that the pressure trigger worked.

In retrospect, either a different camera should have been used, or different software utilized. Originally, the CMU camera was to be used. At the first presentation of the idea for the project, the CMU camera was advised against. If the CMU camera had been used, the imaging may not have been as high a quality as those given in OpenCV, but the implementation would have been more likely to work. The documentation for the CMU camera is better than that for OpenCV. This would have simplified the troubleshooting process, though the end result might not have performed to expectation. Another student in the class had success using program code written in MATLAB to perform the blob tracking from a similar camera. This may have been a viable solution had it been discovered earlier in the course of the project. Alternate solutions were not pursued because each issue up to the last instance with the communications port was resolved. It is possible that the project could still work as designed, given enough time to debug the communications port problem.

The only portion of the original design not built was a power monitoring circuit. Troubleshooting the camera took far more time than expected, so this element was eliminated to get the other components finished. If the robot were to actually be used in the field, then power monitoring would be essential. When the battery approached depletion, the robot would simply enter into the original payload drop-off routine. Once the basket was emptied, the robot would simply stop until the battery was changed or recharged.

Documentation:

Pridgen Vermeer Robotics Xmega128 Manual. Available:

http://plaza.ufl.edu/rhaegar/XMega%20Manual.pdf

AVR XMEGA A1 Device Datasheet. Available:

http://www.atmel.com/dyn/resources/prod_documents/doc8067.pdf

XMEGA A MANUAL. Available:

http://www.atmel.com/dyn/resources/prod_documents/doc8077.pdf

PVR_XMEGA_ Coding. Available:

http://www.mil.ufl.edu/5666/handouts/PVR_XMega_Coding.pdf

HD44780U (LCD-II) datasheet. Available:

http://www.sparkfun.com/datasheets/LCD/HD44780.pdf

Sabertooth 2x12 User's Guide. Available:

http://www.dimensionengineering.com/datasheets/Sabertooth2x12.pdf

Joseph Jones, Bruce Seiger, and Anita Flynn. *Mobile Robots: Inspiration to Implementation*. Natick, MA: A.K. Peters, Ltd., 1999.

United States. Bureau of Naval Personnel. *Basic Electronics*. New York: Courier Dover Publications, 1980.

Bradski & Kaehler. *Learning OpenCV*. Sebastopol, CA: O'Reilly Media, Inc., 2008.

Appendix A: Computer Code

The code presented here is not necessarily the best code to use for such a project. While the sensor routines work fine, there is no real arbitration, and the motor routines have no smoothing functions. Given that the motors rotated extremely slowly, smoothing functions were not really necessary, but good coding practice dictates that they should be there anyway.

```
#include <avr/io.h>
#include "PVR.h"
#include <avr/interrupt.h>
#include <util/delay.h>
#include "print.h"
#include "uart.h"


        int Max = 100;
        int Min = 0;

//--------- Sensor variables

        int CdS;
        int Bumper;
        int SonarLeft;
        int SonarRight;
        int SonarRear;
        int PressSwitch;
        int PT = 3700;              //pressure switch norm @ 3500
        int ST;                     //sonar close at 300
        int i;
//*      char msg;                  //This is reserved for the camera when I get it working



//--------------------------------Main Program--------------------
void main(void)
{
        xmegaInit();                                        //setup XMega
        delayInit();                                        //setup delay functions
        ServoCInit();                                       //setup PORTC Servos
        ServoDInit();                                       //setup PORTD Servos
        ADCAInit();                                             //setup PORTA
analong readings
        lcdInit();                                              //setup LCD on
PORTK
        lcdString("I am the brain of");                 //display message (line 0)
```

```
        lcdGoto(1,0);                                   //move LCD cursor to the
second line (Line 1) of LCD
        lcdString("Robot: Julius");             //display message (line 1)
        PORTQ_DIR |= 0x01;                      //set Q0 (LED) as output
        initUSARTF0(9600);                      //initialize usart for serial comm
        PORTH_DIR |= 0xFF;
        PORTJ_DIR |= 0xFF;

        PMIC.CTRL |= 0x04;
        sei();                                  //interrupts enabled

        delay_ms(200);

        // The paddle engages first
        ServoC4(10);
        delay_ms(100);
        ServoC4(20);
        delay_ms(100);
        ServoC4(40);

        //Then the auger
        ServoC5(20);
        delay_ms(100);
        ServoC5(40);
        delay_ms(100);
        ServoC5(60);
        delay_ms(100);
        ServoC5(80);
        delay_ms(100);
        ServoC5(100);

//------------------------- Routines Called in Order of Implementation
while(1)
{

//---------- Other variables and definitions

SonarLeft = ADCA0();
SonarRight = ADCA1();
SonarRear = ADCA2();
Bumper = ADCA3();
CdS = ADCA4();
PressSwitch = ADCA5();
i=-100;
```

```
        ServoD2(-100);  //gate closed

// ---Code to make LED go blinkity blinkity---

                i+=10;
                if (i >= 0)                     //If on postive half of servo range...
                        PORTQ_OUT = 1;          //  turn on LED
                else                            //If not on positive half of servo range...
                        PORTQ_OUT = 0;          //  turn off LED
                if (i > 100)                    //If past servo range, reset into range
                        i = -100;
                delay_ms(10);                   //delay 10ms



if (Bumper >= 500)
                BumperSenseRoutine(Bumper);

if (SonarLeft <= 1100 || SonarRight <= 1100)
                SonarSenseRoutine(SonarLeft,SonarRight);

if (PressSwitch > PT)
                Dump();

//The Camera routine would normally go here, but the camera is not working

Headlight(CdS);


}

}

// -------------------------Motor Control Behaviors-----------------------------

//These are not particularly well written routines for the motors.
//There are no smoothing functions, but the motors are geared so high that it is not
//really noticable.

//-------------------------------------------------------------------------------

// Cruise Mode for Robot drive until a target is identified
// This makes the robot travel along an arc
```

```
void Cruise(void)
        {
        ServoC0(Max);
        ServoC1(Max-10);
        }

// Turn Left

void LeftTurn(void)
        {
        lcdData(0x01);
        lcdGoto (1,0);
        lcdString("Left");

        ServoC1(Max);
        ServoC0(.5*Max);
        delay_ms(1000);
        }

// Turn Right

void RgtTurn(void)
        {
        lcdData(0x01);
        lcdGoto (1,0);
        lcdString("Right");

        ServoC0(Max);
        ServoC1(.5*Max);
        delay_ms(1000);
        }

// Hard Left Turn

void HardLft(void)

        {
        lcdData(0x01);
        lcdGoto (1,0);
        lcdString("Hard Left");

        ServoC1(Max);
        ServoC0(.5*Max);
        delay_ms(200);
        ServoC1(Max);
        ServoC0(Min);
```

```
        delay_ms(200);
        ServoC1(Max);
        ServoC0(-.5*Max);
        delay_ms(200);
        ServoC1(Max);
        ServoC0(-Max);
        delay_ms(3000);
        ServoC1(Max);
        ServoC0(Min);
        delay_ms(1000);
        }


// Hard Right Turn

void HardRgt(void)
        {
        lcdData(0x01);
        lcdGoto (1,0);
        lcdString("Hard Right");

        ServoC0(Max);
        ServoC1(.5*Max);
        delay_ms(200);
        ServoC0(Max);
        ServoC1(Min);
        delay_ms(200);
        ServoC0(Max);
        ServoC1(-.5*Max);
        delay_ms(200);
        ServoC0(Max);
        ServoC1(-Max);
        delay_ms(3000);
        ServoC0(Max);
        ServoC1(Min);
        delay_ms(1000);
        }

// Reverse Drive

void DriveReverse(SonarRear)
        {

        if(SonarRear < 270)     // in case someone is standing behind robot
                PORTJ_OUT = 0xFF;           // buzzer sounds
        else
```

```
                PORTJ_OUT= 0x00;                    // buzzer off

        lcdData(0x01);
        lcdGoto (1,0);
        lcdString("Reverse");


                ServoC0(-Max);
                ServoC1(-Max);
                delay_ms(6000);

                PORTJ_OUT= 0x00;  //Turn off the buzzer

        }


//---------------Bumper Tests - Avoidance Behavior should Sonar Fail------------

void BumperSenseRoutine(Bumper)
{
int ST = 350;

if (3200>Bumper && Bumper>=2300)        //Left side, or both left

        HardRgt();

 if (2300>Bumper && Bumper>=2075)       //Both front

        DriveReverse();

 if (2075>Bumper && Bumper>=1720)       //Left front

        HardRgt();

 if (1720>Bumper && Bumper>=500)        //Right front, side, or both right

        HardLft();

}
//----------------------- CdS cell test for headlights--------------------------

void Headlight(Cds)
{

if (CdS < 3000)
```

```
                PORTH_OUT = 0xFF;       //light on
                else
                PORTH_OUT = 0x00;       //light off

}

// --------------------Front Sonar Sensor Test - Obstacle Avoidance----------------

void SonarSenseRoutine(SonarLeft,SonarRight)
{
        int ST = 330;           //sonar threshold

if (SonarLeft < ST && SonarRight < ST)

        DriveReverse();

 if (SonarLeft < ST && SonarRight > ST)

        RgtTurn();

 if (SonarLeft > ST && SonarRight < ST)

        LeftTurn();

else
        lcdData(0x01);
        lcdGoto (1,0);
        lcdString("Cruise");
        Cruise();
}
// -------------Check Camera to see if there is an orange in view-----------------

// The camera won't shake hands with the PVR board
// Without it, I have no data to use to write a routine
//* char msg = readF0();

// ------------------------Check Pressure Switch-------------------------------

void Dump(void)
{
        ServoC0(0);
        ServoC1(0);

        lcdData(0x01);
        lcdGoto (1,0);
```

```
        lcdString("Dump");

        PORTJ_OUT=0xFF;          //beep
        delay_ms(1000);
        PORTJ_OUT=0x00;
        delay_ms(1000);

        PORTJ_OUT=0xFF;          //beep
        delay_ms(1000);
        PORTJ_OUT=0x00;
        delay_ms(1000);

        PORTJ_OUT=0xFF;          //beep
        delay_ms(1000);
        PORTJ_OUT=0x00;
        delay_ms(1000);

        ServoD2(60);          //dump basket
        delay_ms(5000);
        ServoD2(-100);                    //close gate


}
```

The source code and header files for the PVR2 board are readily available from Pridgen-Vermeer, and will be posted on the Google Site for the Robot:Julius project. The USART initialization code was provided by Tim Martin, and will posted on the project website as well.

The OpenCV code is adapted from that used by Matt Morgan in his project, originally written by Cristobal Carnero Linan. The main code is printed below, and all supporting source files and header files will be posted on the project website, with the exception of those relating to serial communication, since they did not work.

```
// Cristóbal Carnero Liñán <grendel.ccl@gmail.com>

#include <iostream>
#include <iomanip>

#if (defined(_WIN32) || defined(__WIN32__) || defined(__TOS_WIN__) ||
defined(__WINDOWS__) || (defined(__APPLE__) & defined(__MACH__)))
#include <cv.h>
#include <highgui.h>

#endif

#include "cvblob.h"
```

```
using namespace cvb;

#include "OpenCV_methods.h"
#include "Serial.h"

int main()
{
  CvTracks tracks;
 // HANDLE run_serial_init() ;//----------------Crane

  cvNamedWindow("Blob Tracking Method", CV_WINDOW_AUTOSIZE);

  CvCapture *capture = cvCaptureFromCAM(0);
  //CvCapture *capture = cvCreateFileCapture("http://192.168.1.229/img/video.mjpeg");

  cvGrabFrame(capture);

  IplImage *img = cvRetrieveFrame(capture);

  CvSize imgSize = cvGetSize(img);

  IplImage *frame = cvCreateImage(imgSize, img->depth, img->nChannels);

  IplConvKernel* morphKernel = cvCreateStructuringElementEx(5, 5, 1, 1, CV_SHAPE_RECT,
NULL);

  IplImage* HSVFrame = cvCreateImage(cvGetSize(frame), 8, 3);


  //unsigned int frameNumber = 0;
  unsigned int blobNumber = 0;

  bool quit = false;
  while (!quit&&cvGrabFrame(capture))
  {
    IplImage *img = cvRetrieveFrame(capture);

    cvConvertScale(img, frame, 1, 0);

    //IplImage *segmentated = cvCreateImage(imgSize, 8, 1);

   /* // Detecting red pixels:
    // (This is very slow, use direct access better...)
    for (unsigned int j=0; j<imgSize.height; j++)
      for (unsigned int i=0; i<imgSize.width; i++)
      {
       CvScalar c = cvGet2D(frame, j, i);

       double b = ((double)c.val[0])/255.;
       double g = ((double)c.val[1])/255.;
       double r = ((double)c.val[2])/255.;
       unsigned char f = 255*((r>0.2+g)&&(r>0.2+b));

       cvSet2D(segmentated, j, i, CV_RGB(f, f, f));
      }*/
       IplImage* HSVOut = cvCreateImage(cvGetSize(frame), 8, 1);
       cvConvertImage(frame, HSVOut, 0);
       IplImage* HSVFrame = cvCreateImage(cvGetSize(frame), 8, 3);
```

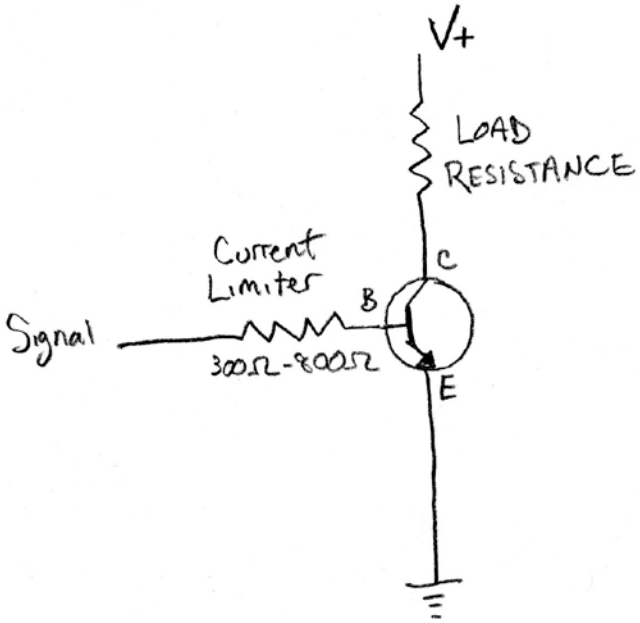
```
        cvConvertImage(frame, HSVFrame, 0);

        cvSmooth(HSVFrame, HSVFrame);
        cvCvtColor(HSVFrame, HSVFrame, CV_BGR2HSV);
        CvFindHSV(HSVFrame, HSVOut, 19, 225, 155, 10, 40, 30);
        cvSmooth(HSVOut, HSVOut);
        cvDilate(HSVOut, HSVOut, 0, 1);
        cvErode(HSVOut, HSVOut, 0, 1);
        cvMorphologyEx(HSVOut, HSVOut, NULL, morphKernel, CV_MOP_CLOSE, 1);
        cvMorphologyEx(HSVOut, HSVOut, NULL, morphKernel, CV_MOP_OPEN, 1);

        cvShowImage("HSV", HSVOut);

    IplImage *labelImg = cvCreateImage(cvGetSize(frame), IPL_DEPTH_LABEL, 1);

    CvBlobs blobs;
    unsigned int result = cvLabel(HSVOut, labelImg, blobs);
    cvFilterByArea(blobs, 1000, 1000000);
    cvRenderBlobs(labelImg, blobs, frame, frame, CV_BLOB_RENDER_BOUNDING_BOX);
    cvUpdateTracks(blobs, tracks, 200., 5);
    cvRenderTracks(tracks, frame, frame,
CV_TRACK_RENDER_ID|CV_TRACK_RENDER_BOUNDING_BOX);

    cvShowImage("Blob Tracking Method", frame);

    /*std::stringstream filename;
    filename << "redobject_" << std::setw(5) << std::setfill('0') << frameNumber <<
".png";
    cvSaveImage(filename.str().c_str(), frame);*/

        //run_serial_init() ;//-------------Crane

    cvReleaseImage(&labelImg);
    cvReleaseImage(&HSVOut);
        cvReleaseImage(&HSVFrame);

    char k = cvWaitKey(30)&0xff;
    switch (k)
    {
      case 27:
      case 'q':
      case 'Q':
        quit = true;
        break;
      case 's':
      case 'S':
        for (CvBlobs::const_iterator it=blobs.begin(); it!=blobs.end(); ++it)
        {
          std::stringstream filename;
          filename << "redobject_blob_" << std::setw(5) << std::setfill('0') <<
blobNumber << ".png";
          cvSaveImageBlob(filename.str().c_str(), img, it->second);
          blobNumber++;

          std::cout << filename.str() << " saved!" << std::endl;
        }
        break;
    }
```

```
    cvReleaseBlobs(blobs);

   // frameNumber++;

  }

  //cvReleaseStructuringElement(&morphKernel);
  cvReleaseImage(&frame);


  //cvDestroyWindow("Blob Tracking Method");
  cvDestroyAllWindows();

  return 0;
}
```

Appendix B: Useful Circuits

The resistor network and voltage divider circuits are presented in the book *Mobile Robot: Inspiration to Implementation.* These are illustrated on pages 139 and 122, respectively. The key to making the resistor network function properly is to significantly change the value of the first resistor in the chain from all of the other resistors in the network. This is what indicates which switch is closed.

The transistor is a simple device to use. NPN transistors were used in this project to send signals to the headlight and buzzer. The simplest configuration is presented in the diagram below. Remember to set the port direction to output in the microcontroller code as part of the initialization procedures.



The load resistance is simply the device that you are connecting (i.e LED array, buzzer). The current limiting resistor value will depend on the transistor that you use, and the desired gain. A value in the neighborhood of 500 Ohms should work for most applications.