Final Report
Michael Mulet
Scaredy Cat
EEL 5666 Intelligent Machines Design Lab
Instructors:
Dr. Arroyo & Dr. Schwartz
Teaching Assistants:
Mr. Weaver & Mr Chilton
Table of Contents:

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>3</td>
</tr>
<tr>
<td>Executive Summary</td>
<td>3</td>
</tr>
<tr>
<td>Introduction</td>
<td>4</td>
</tr>
<tr>
<td>Integrated Systems</td>
<td>5</td>
</tr>
<tr>
<td>Mobile Platform</td>
<td>7</td>
</tr>
<tr>
<td>Actuation</td>
<td>11</td>
</tr>
<tr>
<td>Sensors</td>
<td>12</td>
</tr>
<tr>
<td>Behaviors</td>
<td>13</td>
</tr>
<tr>
<td>Conclusion</td>
<td>16</td>
</tr>
<tr>
<td>References</td>
<td>16</td>
</tr>
<tr>
<td>Appendices</td>
<td>17</td>
</tr>
<tr>
<td>BeagleBoard code</td>
<td>17</td>
</tr>
<tr>
<td>Sensor MSP430</td>
<td>20</td>
</tr>
<tr>
<td>Motor MSP430</td>
<td>24</td>
</tr>
<tr>
<td>Circuit Diagram</td>
<td>31</td>
</tr>
</tbody>
</table>
Abstract:
Scaredy Cat robot is a human interactive game-playing robot that runs away from a scary object. It is cube shaped and its outside is covered with fur, ears and a tail. It is completely autonomous and requires no outside input once the user presses the two power switches. This is possible due to the embedded board, the beagleboar-xM used for high-level processing. It will inspire human emotions through its movements.

Executive Summary:
The robot, a cube shaped cat, runs away from scary object that it "sees". After powering on, the robot waits for the beagleboard-xm, the onboard embedded processor running Ubuntu Linux, to boot up and start its main program. Once it has booted, the beagleboard-xm will send a signal through buffer logic to the sensor MSP430 Microcontroller, which in turn sends a signal to the motor MSP430 Microcontroller. The motor MSP430 sends PWM signals to the motor driver that sends signals to the two DC brushed motors connected to two wheels that start spinning.

The robot will move forward while the sensor MSP430 reads input from three infrared proximity detectors, while the motor MSP430 reads input from two bump switches. If a bump switch is hit or the sensor MSP430 detects that an object is close enough to the proximity detectors, the robot will stop, backup for a random amount of time, then turn for a random duration to avoid the object. The robot can also detect objects while trying to avoid objects so that the robot can maneuver its way around objects.

The onboard webcam connects to the beagleboard-xm, which uses image processing to detect a scary object, the color pink. When the beagleboard detects a pink color robot will spin randomly to the left or the right for a random duration and run away. While this is going on the infrared proximity detectors and the bump sensors will combine to perform object avoidance. This will continue until the
user, detected by the accelerometer, picks up the robot. The robot will re-enter its first mode, the happy mode, and this process will repeat.

**Introduction:**

Scaredy Cat is a human interactive game-playing robot. The goal of the design is create a robot that people in which people will fall in love. The emphasis of Scaredy Cat is human empathy inspired by the robots display of virtual emotion.

This paper specifies the main components of the robot and their functions, it is organized into three sections: the heart, the body, and the mind. It begins with a discussion of the robots heart, integrated system of how all the components fit together, then it moves to the body, Mobile Platform and the Actuators discussing how the robot is built and how it moves. Finally, the paper discusses the robot's mind: how the robot senses the outside world and how it reacts.
Integrated System

USB Webcam

BeagleBoard

Accelerometer

One or more low level microcontrollers

Bump sensors

Infrared proximity sensors

Motor Driver

Wheels

Figure 1
The robot is mounted on cube like platform with two motorized wheels in the front used for locomotion and two rollerballs in the back used for support. The robot has a box for a body, on the front of the box is a webcam, three infrared proximity detectors, and two bump sensors. On top are two cat-like ears. The outside is covered by soft fluffy fur type material, black velvet.

Inside the box will be the electronics, the microprocessor board, beagleboard -xm[3], is complemented with msp430[2] microcontrollers. The webcam is connect to the beagleboard by USB. Inputs producing analog voltage, such as the accelerometer, interface to the msp430 which in turn interface to the beagleboard. The communications are performed by outputting and setting GPIO pins.
If I could do it again, I would use serial communication, most likely SPI, because it would have been easier to wire, and I would have had more free pins for more functions.

**Mobile Platform:**

![Complete Solidworks Model](image)

*Figure 3 Complete Solidworks Model*
Figure 4 Front of Mobile Platform

Figure 5 Sides of the Mobile Platform
Figure 6 The Back of the Mobile Platform

Figure 7 The Bottom of the Mobile Platform
The beagleboard xm is mounted on the left side of the robot. It is attached by hook and loop sticker. The MSP430 boards are resting at the bottom of the robot. The platform is covered by black velvet to simulate fur, and a tail and two ears are glued on to the back and top respectively. It looks like a cube cat.

If I could do it again, I would have made the platform design easier to assemble and reassemble by adding hinges. This way I could easily take the robot apart to work on the electronics inside. The way it is now, working with the board while it is mounted on the platform is hard because the space is really small and it is hard bring in enough light to see.
Figure 9

Two brushed DC gear-motors drive the robot. The motors have a stall torque of 30 oz-in and
320 rpm at free run. In order to drive the motors, the MSP430 connects to a DRV8835 Motor Driver, the motor driver accepts a PWM input from the msp430 and a phase input that determines the direction that the motor will rotate. The duty cycle of the PWM determines the speed of the motor. In walk mode each motor is driven by a PWM input of 25% duty cycle, in run mode it uses a PWM of 50% duty cycle. While spinning the fast wheel spins takes a PWM input of 75% duty cycle.

I used these motor driver and motor combination because they were small, lightweight, and easy to use. I originally had less powerful motors, 12 oz-in with 120 rpm, but they did not produce enough speed. It looked like the robot was crawling in happy mode, and walking in scared mode. With the more powerful motors, the robot walks in happy mode, and runs in scared mode.

**Sensors:**

- **Bump Sensors:**
  
  ![Figure 10](image)

  - The bump sensors are switches [3] mounted on the front of the platform. When the switches are depressed, the motor MSP430 reads the value in through GPIO and activates the avoid routine.

- **Infrared proximity detectors [4]:**
Figure 11

- Used for proximity detection. Produces an analog voltage, which is read by the sensor MSP430 through an ADC. If the analog voltage is higher than a threshold, an object is detected. The sensor MSP430 sends a signal to the motor MSP430 through GPIO. Then the motor control begins the avoid routine.

Figure 12

- Accelerometer[5]:
  - Accelerometer will be used to sense tilt, when the robot is tilted it will send a signal to the motor MSP430, which cause the robot to leave scared mode if in scared mode.
  - If the robot is turning or in happy mode, then there is no change.

- Webcam:
Used to detect the color pink which causes the robot to enter scared mode. The camera connects to the beagleboard which uses openCV for image processing. The program captures a frame from the camera, downsamples it, converts the frame from RGB color mode to HSL color mode, then thresholds the image in a certain range of Hue such that all pixels considered "pink" will be white and everything else will be black. Then the program calculates the mean of the image, and if the mean is above a certain threshold then the beagleboard sends a signal to the MSP430s.

A large portion of my time this semester was consumed by the beagleboard. I had never worked with an embedded system before, and I had very little experience with Linux. The least I can say is that I had a tumultuous time trying to get the system to work.

The beagleboard shipped with a lightweight Linux distribution called Angstrom. It was rather easy to use, and I was able to get openCV to make and run, but I ran into an issue with the webcam, the OS did not have any drivers to support my webcam. My next option was to write my own driver or switch operating systems. I decided to install Ubuntu. The beagleboard-xM boots from an inserted microSD card, so loading Ubuntu was as simple as using the dd utility to save a disc image of Ubuntu onto the microSD. Unfortunately, the image provided by Canonical ran slowly. After searching for different option, I found a minimal image that used the LXDE desktop. This version of ubuntu ran much faster.

The process was smooth from there until I had to make my system compatible with the needs of my robot; ie, it had to do two things upon power up: automatically log on, and start my program automatically. This allows my robot to boot up and start its program without any human intervention; it need only be turned on. Auto login turned out to be an easy issue to fix, but getting the program to autostart was more difficult. It appeared as if the program was working, I could see the output of the camera after it had done its processing, but it could not communicate with the low level processors. It turned out that program needed super user privileges to access the processors GPIO. So my next
problem was that I needed to give the program privileges without going in and typing my password on the keyboard. The solution was to edit a certain file, the sudoers file, to give permission without any passwords, my first attempt to do this ended with locking myself out of the sudo command, a command used to give permissions, and thus locking myself out of the system entirely. Luckily, by that time I had discovered that the dd utility made backups easy, so I could easily restore my system to an earlier state. I found a much safer way to edit the sudoer file, the visudo command, and was on my way.

In the end, the beagleboard works great and I am glad to have learned so much about embedded linux programming.

Behaviors:

The robot will start in a happy mode in which it will slowly walk using collision detection to avoid objects. Then, when the beagleboard detects a pink color, the robot will spin randomly to the left or the right for a random duration and run away. While this is going on the infrared proximity detectors and the bump sensors will combine to perform object avoidance. This will continue until the user, detected by the accelerometer, picks up the robot. The robot will enter happy mode. Repeat.

While programming the behavior of the robot I learned the importance of adding "random" elements to the robot behavior. Let me explain what I mean by adding "random" elements. I used to program specific behaviors for specific instances, for example: when a collision was detected at the robots right side, the robot would stop for a half second, back up for 2 seconds then turn left for 2 seconds. However, this became troublesome as sometimes the robot would collide with another object while turning, and I had to make a new function for colliding while turning as I wanted to make sure the robot would not get trapped. In the end, my efforts hard code obstacle avoidance pushed over the microcontroller's memory capacity. So, I decided to take a break from that problem and move on to programming code for my sensor. Unfortunately, I noticed that I had overwritten both my behavior
code, and the code's backup. I took that opportunity to try an entirely new approach to my obstacle avoidance. Instead of backup up and turning at set intervals, I made it so that the program would read values from the least significant byte of the timer (the timer used to run PWM to the motors). Now, the robot would back up and turn randomly, and furthermore, I made it so that if the robot encountered a collision while turning, the same function would be called recursively, this time with new random values, and this could repeat virtually forever. Now the robot is significantly better at avoiding objects, and it looks more life-like too.

Conclusion:

The ScaredyCat robot is a robot that uses image processing to detect scary objects, the color pink, then communicates this scared state to two low level microcontrollers that are using sensors to read in input for collision detection, and communicate to a motor driver which drives two motors. The robot has obstacle avoidance routines and a second set of behaviors for when the robot is "scared".

If I had more time I would have added a microphone array on top of the robot that would allow the robot to perform sound localization. This way the robot would be able to run away from loud sounds that it "hears" instead of just scary things that it "sees". This was my first idea, and I spent a lot of time trying to get sound localization to work, only for it to end in failure. If I could start the project over, I would have gone with the image processing, instead of the sound localization, in the first place and spent the time I used working of sound localization to add more human interactivity. For starters, I would have added a LCD screen face so that the robot would display emotion, and I would have added arms that spin around when it is scared. Overall, while I can see all the ways I can improve ScaredyCat in the future, I am still proud of how it is now.

References:


Appendices:

Program Code:
Beagleboard code

```
//test of open CV, Michael Mulet 2013
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
using namespace cv;
using namespace std;
FILE *fp;

int main()
{
    //capturing video form camera0
    CvCapture* capture;
    Mat frame,dst,RHframe,tmp;
    capture = cvCaptureFromCAM(0);
    //return an error if capture was unsuccessful
    if(!capture){
        printf("Failed to capture from camera0");
        return -1;
    }
    //namedWindow("camera", CV_WINDOW_AUTOSIZE);
```
namedWindow("threshold", CV_WINDOW_AUTOSIZE);

char set_value[4];
if((fp = fopen("/sys/class/gpio/export", "ab")) == NULL){
    printf("Failed to open export");
    return -1;
}

rewind(fp);
strcpy(set_value,"130");
fwrite(&set_value,sizeof(char),3,fp);
fclose(fp);

if((fp = fopen("/sys/class/gpio/gpio130/direction", "rb+")) == NULL){
    printf("Failed to open direction");
    return -1;
}

rewind(fp);
strcpy(set_value,"out");
fwrite(&set_value,sizeof(char),3,fp);
fclose(fp);

if((fp = fopen("/sys/class/gpio/gpio130/value", "rb+")) == NULL){
    printf("Failed to open value");
    return -1;
}

rewind(fp);
strcpy(set_value,"0");
fwrite(&set_value,sizeof(char),1,fp);
fclose(fp);

if((fp = fopen("/sys/class/gpio/export", "ab")) == NULL){
    printf("Failed to open export");
    return -1;
}

rewind(fp);
strcpy(set_value,"127");
fwrite(&set_value,sizeof(char),3,fp);
fclose(fp);

if((fp = fopen("/sys/class/gpio/gpio127/direction", "rb+")) == NULL){
    printf("Failed to open direction");
    return -1;
}

rewind(fp);
strcpy(set_value,"out");
fwrite(&set_value,sizeof(char),3,fp);
fclose(fp);

if((fp = fopen("/sys/class/gpio/gpio127/value", "rb+")) == NULL){

printf("Failed to open value");
    return -1;
}

//send the ready signal to the msp430s
rewind(fp);
strcpy(set_value,"0");
fwrite(&set_value,sizeof(char),1,fp);
fclose(fp);

while(1){
    //create a frame from the camera
    IplImage* iplImg = cvQueryFrame(capture);
    frame = iplImg;
    //return an error if frame is empty
    if(frame.empty()){
        printf("Failed to Query Frame from capture");
        return -1;
    }
    pyrDown(frame,tmp);
    pyrDown(tmp,frame);
    //convert from RBG to HSV
cvtColor(frame, RHframe, CV_BGR2HSV, 0);
    //Return 1 if pixel is in specified range (is pink)
inRange(RHframe, Scalar(130,100,100),Scalar(190,255,255), dst);

    imshow("threshold",dst);
    //take the mean of the image
    double avg = mean(dst)[0];

    if((fp = fopen( "/sys/class/gpio/gpio130/value", "rb+")) == NULL){
        printf("Failed to open value");
        return -1;
    }
    //if the mean is greater than 9.5 pink is detected send the signal to the
    msp430s
    if(avg > 9.5){
        cout << "1\n";
        rewind(fp);
        strcpy(set_value,"1");
        fwrite(&set_value,sizeof(char),1,fp);
        fclose(fp);
    }
    else{
        cout << "0\n";
        rewind(fp);
        strcpy(set_value,"0");
        fwrite(&set_value,sizeof(char),1,fp);
        fclose(fp);
    }
}

}//end while loop
}//end program
Sensor MSP430

#include "io430.h"

//global variables and constants
unsigned int frontLeftIr[2];
unsigned int frontLeftIrIndex = 0;
unsigned int frontLeftIrAvg;

unsigned int frontRightIr[2];
unsigned int frontRightIrIndex = 1;
unsigned int frontRightIrAvg;

unsigned int frontCenterIr[2];
unsigned int frontCenterIrIndex = 1;
unsigned int frontCenterIrAvg;

unsigned int backLeftIr[2];
unsigned int backLeftIrIndex = 0;
unsigned int backLeftIrAvg;

unsigned int backRightIr[2];
unsigned int backRightIrIndex = 1;
unsigned int backRightIrAvg;

//begin function definitions

void init_adc(){
  //set up ADC
  ADC10AE0 = 0x4F; //to select A0,A1, and A2,A3,and A6
  ADC10CTL1 = 0x0018; //setup ADC,select A0 as input
  ADC10CTL0 = 0x0010;//turn on reference generator, select Vcc and Vss as
                     //upper and lower reference, sample at 4*ADC10CLKs
}

void init_gpio(){
  /*
   P1DIR |= 0x40; //P1.6 output
   P1OUT &= 0xBF; //P1.6 out 0
   P2DIR &= 0xDE; // bump switches are inputs
   P2DIR |= 0x14; //P2.4 output,P2.2 output
   P2SEL |= 0x14; //P2.2 Timer 1 A1 out, P2.4 Timer 1 A2 out
   */
  //the outputs
  P1DIR |= 0xB8; //P1.3,P1.4,P1.5,,P1.7 output
  P2DIR |= 0x21; //P2.0,2.5 output
  P1OUT &= 0x47; //P1.3,P1.4,P1.5,P1.7 output zero
}
P2OUT &= 0xDE; //P2.0,P2.5 zero

//the inputs
//P2.1 input pinkDetect
//P2.2 input left
//P2.3 INPUT RIGHT
//P2.4 Ready
P2DIR &= 0xE1; //pins 2.1,2.2,2.3,and 2.4 input

unsigned int measure_ir(unsigned int irCode){
    //irCode, match up to the analog input being used
    //so A0 is frontLeftIr
    //A1 is frontRightIr
    //A2 is frontCenterIr
    ADC10CTL0 &= 0xFFFC;//ENC =0 now ADC10CTL1 can be modified
    __delay_cycles(50);
    ADC10CTL1 &=0x0FFF;
    ADC10CTL1 |= (irCode <<12); //choose the Analog input in irCode
    __delay_cycles(50); //delay needed for change to take effect
    ADC10CTL0 |= 0x03; //enable ADC10, also starts conversion
    __delay_cycles(50);
    return ADC10MEM; // read in value of adc
}

unsigned int check_ir(unsigned int ir[2]){ //return 1 if object, 0 if no object
    //unsigned int ir[8]; //array of integers to store the ir values
    unsigned int avg;
    for(int i=0;i<2;i++) { //sum all the inputs
        avg +=ir[i];
    }
    avg = avg >> 1; //shift right by 1 to get the mean
    unsigned int goodCount =0;
    unsigned int goodIr[2]; //the array of good values
    unsigned int k=0; //the array index for goodIr
    unsigned int lowBound = avg-(avg >>2);
    unsigned int highBound = avg+(avg >>2);
    for(int i=0;i<2;i++){
        // if the value is less that 3/4 avg or greater than 5/4 avg, it is bad
        if((ir[i] > lowBound) && (ir[i] < highBound)) {
            goodIr[k]=ir[i];
            k++;
            goodCount++;
        }
    }
    //take new avg
    avg = 0;
    for(int i=0;i<goodCount;i++) { //sum all the inputs
        avg +=goodIr[i];
    }
    avg = avg >> 1; //shift right by 1 to get the mean
    return avg;
}
void init_timer()
{
    TA1CCR0 = 383000; //period used for the new measurement of the IR
    TA1CCTL1 = 0x0000; //set timer 1 to compare to number in the period
    TA1CTL = TASSEL_2 + MC_1 + TAIE; // Start the timer, select SMCLK as clk source
}

int main( void )
{
    // Stop watchdog timer to prevent time out reset
    WDTCTL = WDTPW + WDTHOLD;

    //IR has a new measurement every 38 + or - 9.6 ms

    init_adc();
    init_gpio();
    init_timer();

    while(1){
        if(TA1CTL & 0x01 == 1) { //interrupt pending
            TA1CTL &= 0xFFFE; //clear the flag
            frontLeftIr[frontLeftIrIndex] = measure_ir(0);
            frontLeftIrIndex++;
            if(frontLeftIrIndex == 2){
                frontLeftIrIndex=0;
                frontLeftIrAvg = check_ir(frontLeftIr);
            }
            frontRightIr[frontRightIrIndex] = measure_ir(1);
            frontRightIrIndex++;
            if(frontRightIrIndex == 2){
                frontRightIrIndex=0;
                frontRightIrAvg = check_ir(frontRightIr);
            }
            frontCenterIr[frontCenterIrIndex] = measure_ir(2);
            frontCenterIrIndex++;
            if(frontCenterIrIndex == 2){
                frontCenterIrIndex=0;
                frontCenterIrAvg = check_ir(frontCenterIr);
            }
            backLeftIr[backLeftIrIndex] = measure_ir(3);
            backLeftIrIndex++;
            if(backLeftIrIndex == 2){
                backLeftIrIndex=0;
                backLeftIrAvg = check_ir(backLeftIr);
            }
            backRightIr[backRightIrIndex] = measure_ir(6);
            backRightIrIndex++;
            if(backRightIrIndex == 2){

backRightIrIndex = 0;
backRightIrAvg = check_ir(backRightIr);

}

if ((frontLeftIrAvg > 682)) {
    // Pin2.0 output
    P2OUT |= 0x01;
} else {
    P2OUT &= 0xFE;
}

if ((frontRightIrAvg > 682)) {
    P2OUT |= 0x20;
} else {
    P2OUT &= 0xDF;
}

if ((frontCenterIrAvg > 682)) {
    // using value of clock to make a "random" value
    // it will "randomly" turn left or right
    if ((TA1R & 0x0001) != 0) {
        P2OUT |= 0x01;
    } else {
        P2OUT |= 0x20;
    }
}

if ((backLeftIrAvg > 682)) {
    P1OUT |= 0x10;
} else {
    P1OUT &= 0xEF;
}

if ((backRightIrAvg > 682)) {
    P1OUT |= 0x80;
} else {
    P1OUT &= 0x7F;
}

// communication from the beagleboard
if (((P2IN & 0x10) == 0)) {
    // ready
    P1OUT |= 0x08;
    if (((P2IN & 0x02) == 0)) {
        // pinkDetected
        // Pin1.5 output
        P1OUT |= 0x20;
    } else {
        P1OUT &= 0xDF;
    }
} else {
    P1OUT &= 0xF7;
void rightmotor(int speed, int direction){
    if(speed > period){ //prevent malignant input values
        }
    else{
        TA1CCR2 = speed; //right motor duty cycle
        if(direction == 0){
            P2OUT &= 0xF7; //backwards
        }
        else{
            P2OUT |= 0x8; //forward
        }
    }
}

void leftmotor(int speed, int direction){
    if(speed > period){ //prevent malignant input values
        }
    else{
        TA1CCR1 = speed; //right motor duty cycle
        if(direction == 0){
            P2OUT &= 0xFD; //backwards
        }
        else{
            P2OUT |= 0x2; //forward
        }
    }
}
void init_adc(){
    //set up ADC
    ADC10AE0 |= 0x30; //to select A4,A5
    ADC10CTL1 = 0x4018; //setup ADC,select A4 as input
    ADC10CTL0 = 0x0010; //turn on reference generator, select Vcc and Vss as
    //upper and lower reference, samle at 4*ADC10CLKs
}

void init_pwm(){
    TA1CCR0 = period; //period used for the PWM
    TA1CCCTL1 = OUTMOD_7; //set reset mode, used for PWM
    TA1CCCR1 = period / 2; // left motor speed, duty cycle, 0= 0%, period = 100%
    //controls compare A1 register, in OUTMOD_7, the output
    // is set to 1 when the timer restarts, and is set
    // to 0 when TAR = TA1CCR1
    TA1CCCTL2 = OUTMOD_7; // same as TA1CCR1 except everythin is 2 instead of 1
    TA1CCCR2 = period / 2; //right motor speed
    TA1CTL = TASSEL_2 + MC_1; // Start the timer, select SMCLK as clk source
}

void init_gpio(){
    //pin 1.0 input front right IR
    //pin 1.1 input front left IR
    //pin 1.2 input pink detectd
    //pin 1.3 input back left IR
    //pin 1.4 adc accelerometer adc
    //pin 1.5 adc accelerometer adc
    //pin 2.0 input right bumper
    //pin 2.1 output motor
    //pin 2.2 output motor
    //pin 2.3 output motor
    //pin 2.4 output motor
    //pin 2.5 input left bumper
    //pin 1.6 input back right IR
    //Pin2.6 output statas led
    //pin2.7 output status led
    P1DIR &= 0xB0; //pinl 1.0,1.1,1.2,1.3,1.6 input
    P2DIR &= 0xDE; //2.0, 2.5 bump switches are inputs
    P2DIR |= 0xDE; //P2.4 output,P2.2 output
    P2OUT = 0x00;
    P2SEL &= 0x3F;
    P2SEL |= 0x14; //P2.2 Timer 1 A1 out, P2.4 Timer 1 A2 out
}

int check_ir(int ir){ //return 1 if object, 0 if no object
    if(ir == 0){ //check front left
        if((P1IN & 0x02) != 0){
            return 1;
        }
        else{
            return 0;
        }
    }
}

if(ir == 1){ //check frontRight
    if((P1IN & 0x01) != 0){
        return 1;
    }
    else{
        return 0;
    }
}
if(ir == 2) { //check backleft
    if((P1IN & 0x08) != 0){
        return 1;
    }
    else{
        return 0;
    }
}
if(ir == 3){ //check backRight
    if((P2IN & 0x40) != 0){
        return 1;
    }
    else{
        return 0;
    }
}
return 0;
}

int check_rb(){ //return 1 if right bumper hits an object, 0 if no hit
    if((P2IN & 0x01) != 0){ // right bumper hit
        return 1;
    }
    else{
        return 0;
    }
}

int check_lb(){
    if((P2IN & 0x20) != 0){ //return 0 if right bumper hits an object, 0 if no hit
        return 1;
    }
    else{
        return 0;
    }
}

void wait(int time,int direction); //forward declartion

int check_collision(int direction); // another forward declaration

void avoid(int direction){ //direction is 0 turn left, direction is 1 turn right
    //stop motors
    rightmotor(0,forwards);
    leftmotor(0,forwards);
    __delay_cycles(500000); //delay half a second
    //go backwards
    rightmotor(period/2,backwards);
leftmotor(period/2, backwards);
int randTime = (TAIR & 0x001F);
if(randTime < 4){
  randTime = 4;
}
// for pseudo-random amount of time from .25s to two second
wait(randTime, backwards);
// stop
rightmotor(0, forwards);
leftmotor(0, forwards);
__delay_cycles(500000); // delay half a second
if(direction != 0){ //
  rightmotor(period*.5, forwards);
  leftmotor(period/4, forwards); // turn right
} else{
  leftmotor(period*.5, forwards);
  rightmotor(period/4, forwards); // turn left
}
randTime = (TAIR & 0x001F);
if(randTime < 4){
  randTime = 4;
}
// for pseudo-random amount of time from .25s to two second
wait(randTime, forwards);

unsigned int measure_axis(int axis){ // measure input from accelerometer
  ADC10CTL0 &= 0xFFFC; // ENC = 0 now ADC10CTL1 can be modified
  __delay_cycles(50);
  ADC10CTL1 &= 0x0FFF;
  ADC10CTL1 |= (axis << 12); // choose the Analog input in irCode
  __delay_cycles(50); // delay needed for change to take effect
  ADC10CTL0 |= 0x03; // enable ADC10, also starts conversion
  __delay_cycles(50);
  return ADC10MEM; // read in value of adc
}

void wait(int time, int direction){ // wait and also check for collision
  TA0CCR0 = 0xFFFF; // max period
  TA0CTL = TASSEL_2 + MC_1;
  unsigned int hitFlag = 0;
  for(int i=0; i < time; i++){ // two second wait
    while((TA0CCTL0 & 0x01) == 0){
      // polling Timer flag
      hitFlag = check_collision(direction);
      if(hitFlag == 1){
        break;
      }
    }
    TA0CCTL0 &= 0xFFFE; // clear the timer
  }
  break;
}

TA0CTL = MC_0; // stop timer
int check_collision(int direction){
    if (direction == 1) {
        if ((check_rb() == 1) || (check_ir(frontRight) == 1)) {
            P2OUT &= 0x3F;
            P2OUT |= 0x40;
            avoid(left);
            return 1;
        }
        if ((check_lb() == 1) || (check_ir(frontLeft) == 1)) {
            P2OUT &= 0x3F;
            P2OUT |= 0x80;
            avoid(right);
            return 1;
        }
    } else {
        if (((P1IN & 0x08) != 0) || ((P1IN & 0x40) != 0)) {
            return 1;
        }
    }
    return 0;
}

int scared_mode() {
    P2OUT |= 0xC0;
    // stop motors
    rightmotor(0, forwards);
    leftmotor(0, forwards);
    __delay_cycles(500000); // delay half a second
    // turn around
    if ((TA1R & 0x0001) != 0) { // pseudo random turn around left or right
        rightmotor(period/4, backwards);
        leftmotor(period*3/4, forwards);
    } else {
        rightmotor(period*3/4, forwards);
        leftmotor(period/4, backwards);
    }
    int randTime = (TA1R & 0x001F);
    if (randTime < 4) {
        randTime = 4;
    }
    // for pseudo-random amount of time from .25s to two second
    wait(30, forwards);
    rightmotor(0, forwards);
    leftmotor(0, forwards);
    __delay_cycles(100000); // delay half a second
    rightmotor(period/2, forwards);
    leftmotor(period/2, forwards);
    signed int yCharge = 0;
    signed int xCharge = 0;
    while (!) {
        // check if any bumper or IR has been hit
        P2OUT |= 0xC0;
if (check_collision(forwards) == 1) {  
    return 1;
}

unsigned int yAxis = measure_axis(y);  
if (yAxis < 350) {  
    yCharge--;  
}  
if (yAxis > 650) {  
    yCharge++;  
}  
if ( (yCharge > 50) || (yCharge < -50)) {  
    return 1;
}

unsigned int xAxis = measure_axis(x);  
if (xAxis < 350) {  
    xCharge--;  
}  
if (xAxis > 650) {  
    xCharge++;  
}  
if ( (xCharge > 50) || (xCharge < -50)) {  
    return 1;
}

}

int main(void)  
{
    // Stop watchdog timer to prevent time out reset  
    WDTCTL = WDTPW + WDTHOLD;

    init_gpio();  
    init_pwm();  
    init_adc();  
    unsigned int adc = 0; // will store ADC10MEM value

    // left direction  
P2DIR |= 0x2; // P2.1, controls phase, forwards or backwankards, of left motor  
P2OUT |= 0x2; // forward

    // right direction  
P2DIR |= 0x8; // P2.7, controls phase, forwards or backwankards, of right motor  
P2OUT |= 0x8; // forward

    // start speed.  
    rightmotor(period/4, forwards);  
    leftmotor(period/4, forwards);

    // setup and start Timer A0  
    TA0CCR0 = 0xFFFF; // max period  
    TA0CTL = TASSEL_2 + MC_1;  
    for (int i=0; i< 30; i++) {  // two second wait for transients
        // poll timer flag  
        while((TA0CCTL0 & 0x01) == 0){
TA0CCTL0 &= 0xFFFF; //clear the timer flag

TA0CTL = MC_0; //stop timer

while(1){

    P2OUT &= 0x3F;
    rightmotor(period/4, forwards);
    leftmotor(period/4, forwards);
    if(check_collision(forwards) == 1){
        P2OUT &= 0x3F;
    }
    if((P1IN & 0x04) != 0){ //if pink is detected
        scared_mode();
    }
} //end of while loop
return 0;

Circuit Schematic:
Figure 13
Voltage regulator circuits from [7] [8]