University of Florida
Department of Electrical and Computer Engineering
Intelligent Machine Design Laboratory
EEL 5666 Fall 2012

Fallen Angel

Daniel Z. Frank

Instructors: Drs. Antonio Arroyo and Eric Schwartz
Teaching Assistants: Timothy Martin and Joshua Weaver

Contents

I. Abstract

Fallen Angel is an autonomous mobile robot that can act as a personal emergency response system. Fallen Angel patrols the home avoiding obstacles while searching for someone who has fallen. Once a person is detected lying on the floor, it calls for help.

II. Executive Summary

Fallen Angel is designed to detect when someone has fallen on the floor and then call for help. It navigates the home, searching for someone, while engaging in basic obstacle avoidance by using an array of infrared proximity sensors and an ultrasonic range finder. The IP camera mounted on the front of the robot sends its raw video feed to a laptop running OpenCV. The OpenCV program detects faces in real time by searching for Haar-like features using the Viola-Jones method [8]. When a face is detected, the laptop sends a message to the robot via Bluetooth to stop driving and keep the camera fixed on the person.

Once a person is detected, OpenCV launches a Visual Basic SMTP application that sends a prewritten message to a predefined list of e-mail accounts. The message is also sent to a predefined list of phone numbers via a text message. Once the message is sent, an audial confirmation is given using a text-to-speech algorithm. In the e-mail, a photo of the fallen person is attached. This allows the recipients to verify the authenticity of the emergency.

III. Introduction

For the elderly, living by oneself can be a difficult task. As someone becomes older, they statistically become more prone to falling and injuring themselves. The Center for Disease Control and Prevention (CDC) states that one out of three adults ages 65 and older fall every year. [1] Due to this need, many products are now available for these people and their families to gain some peace of mind. They are known as personal emergency response systems (PERS).

A personal emergency response system allows someone who is frail, elderly, or has other disabilities to easily contact emergency services. The PERS consists of a radio transmitter, a console connected to a landline phone, and an emergency response center to monitor calls. The transmitter is usually the same shape and size of a small cell phone. It can be worn on a belt, around the neck, or carried in a pocket. When an emergency occurs, the user holds down the button on the transmitter which signals the console on the phone to dial one or more emergency numbers. However, these devices tend to be expensive. The hardware can cost anywhere from $200 to $1500. Additionally, there is typically a monthly fee between $10 and $30. [2] These devices are also invasive, in the sense that the user must carry the radio transmitter around with them at all times while at home.

Fallen Angel is a mobile autonomous robot that can search a home for a human that is lying on the ground. Once it locates a person, it can function as a PERS by notifying emergency services as well as family, neighbors, and anyone else that it was preprogrammed to contact. The

advantage of Fallen Angel to a traditional PERS is that it can be activated when the owner of the system is unable to call for help. It also does not require the owner to constantly carry around a transmitter. For even greater safety, Fallen Angel can be used in conjunction with a traditional PERS in order to ensure the greatest chance for a successful rescue.

IV. Integrated System

The IP camera directly communicates with a router connected to the laptop computer via a wireless connection. There, the raw camera feed is sent to an OpenCV program for analysis. The program checks the image to see if it detects a face by using a Haar-like features training algorithm. The information being generated from the OpenCV program is sent to the BlueSMiRF via Bluetooth. The BlueSMiRF has a direct connection to the Epiphany DIY board. The board has code that can read the information from the laptop and process it. The board is also directly connected to the ultrasonic range finder and the infrared sensors. The board uses the information gathered by the sensors and the laptop to appropriately control the various actuators. During the search mode and obstacle avoidance behavior, the Epiphany DIY uses the proximity sensors to avoid obstacles while using the IP camera to search for a fallen human. Once a human is spotted, the laptop acts as a PERS and begins calling for help.

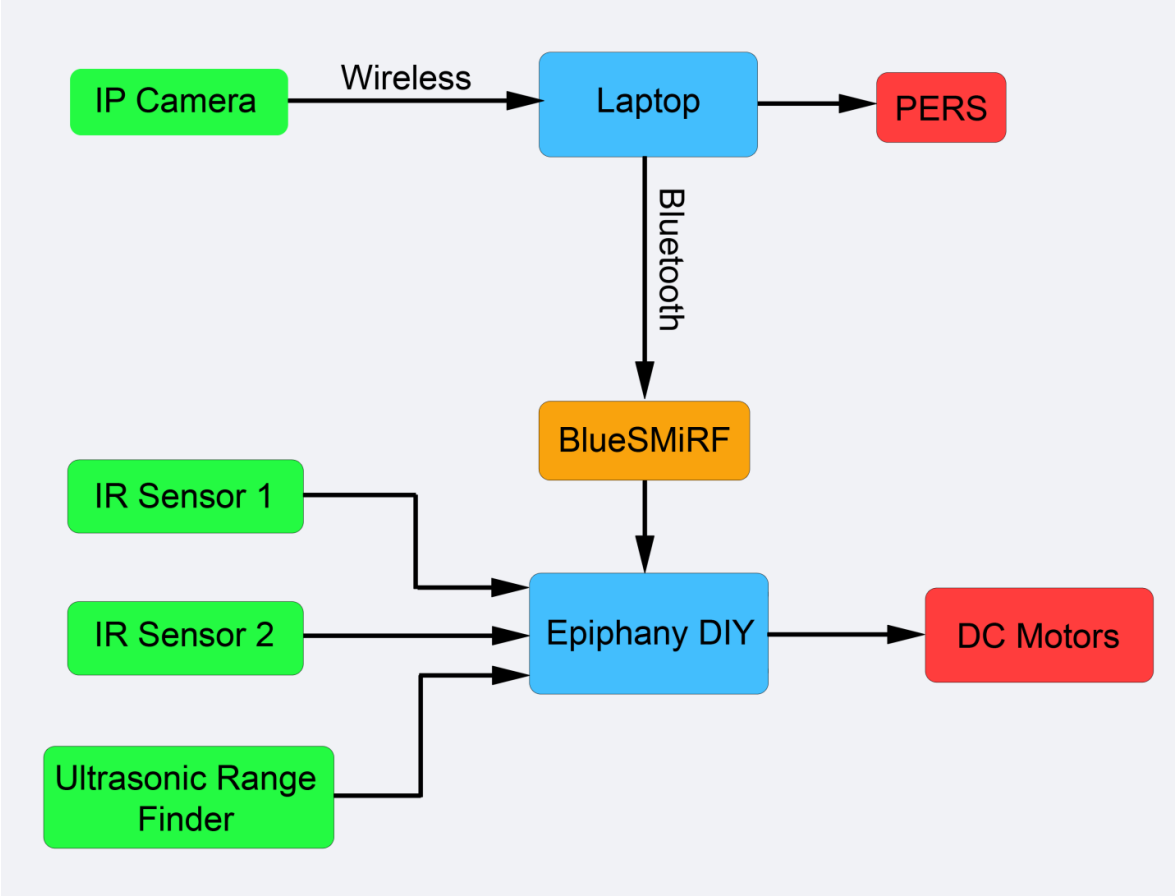

Fig. 1 Functional Block Diagram

V. Mobile Platform

The mobile platform is a simple yet functional design. It features a two-wheel drive system actuated by small DC motors. The platform consists of two main levels. The bottom level rests above the housing for the motors and provides space for the batteries. In addition, a bracket is attached to the front of the bottom level and serves as the mount for the IP camera. The top level is where all of the sensors and major electrical components are located. The Sharp IR sensors are mounted to the front of the top level via brackets. The ultrasonic range finder is mounted onto the base of the IP camera. The board is attached to the top level as well using plastic spacers. There is also room on the top level for the BlueSMiRF which will be attached with Velcro for ease of attachment and removal.

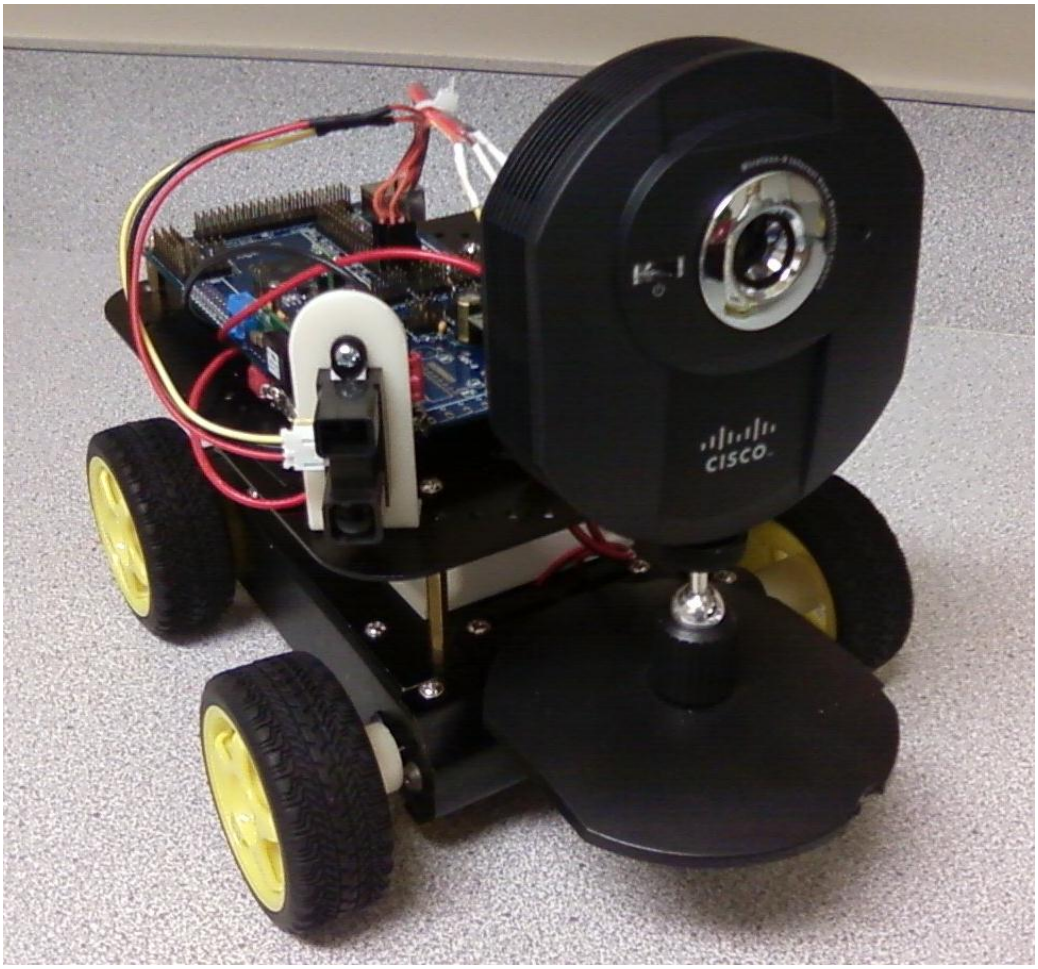

Fig. 2 Photo of the fully assembled robot

Since the application of the robot only requires a simple mobile platform design and basic capabilities, the platform is constructed by modifying an Arduino all-wheel-drive chassis. By modifying a pre-existing frame rather than starting from scratch, it means that less time was spent working on the frame and more time was spent working on the vision processing and computer programming.

As mentioned in the previous paragraph, there were some significant modifications that had to be made to the purchased chassis. First, new motors had to be ordered since the motors that came with the chassis were rated for a maximum of 6 volts. This presented a problem since the output voltage of the motor controllers on the Epiphany DIY can go as high as 12 volts. Since the new motors were not designed to be used in the chassis, a method of mounting the motors to the chassis as well as attaching the motors to the wheels had to be designed. Attaching the motors to the chassis was as simple as drilling some new holes and screwing them together.

The issue of attaching the motor to the wheels was a much more difficult issue. The motor's shaft is D-shaped while the corresponding mate on the wheel is rectangular. To get the two to mate together, I designed an adapter in SolidWorks and printed a couple of copies out of ABS plastic on a 3-D printer. The first time I built them I made the mistake of printing them vertically. I was motivated to build them in this orientation since they would not require any support material. However, the strength of the rapid-prototyped part is directly related to the orientation in which it is made. Building the part in the vertical orientation made it susceptible to shearing. As a result, the part snapped in half while the robot was driving around and I had to drill out the material in the wheel so that I could replace it with another. The figure below depicts one of the wheel adapters alongside with one that has sheared in half.

Fig. 3 On the left is a wheel adapter is still intact and on the right is one that has failed in shear

The second time I built the parts, I did it in the horizontal orientation. Printing it this way did require support material, so I had to put it in a tank full of hot sodium-hydroxide. The solution dissolved the support material while leaving the model material intact. Unfortunately, the parts were so small that they fell into the drain of the tank. It wasn't until the tank had to be drained for maintenance that I was able to retrieve the parts. While these pieces were significantly stronger than the first batch, they still failed after some usage. The final solution was to get aluminum set screw hubs to attach the wheels to the motor.

The next issue I encountered was how to properly mount the DIY Epiphany board and the IR sensors. The board issue was resolved simply by drilling some new holes. For the IR sensors, I found a drawing of the sensor online and used the dimensions to rapid prototype a custom mount. Finally, in order to keep the battery from moving while the robot was in operation, I took dimensions of the battery using calipers and I made a rapid prototype for a holder to secure the battery in place. The rapid prototype of the battery holder can be seen in the figure below.

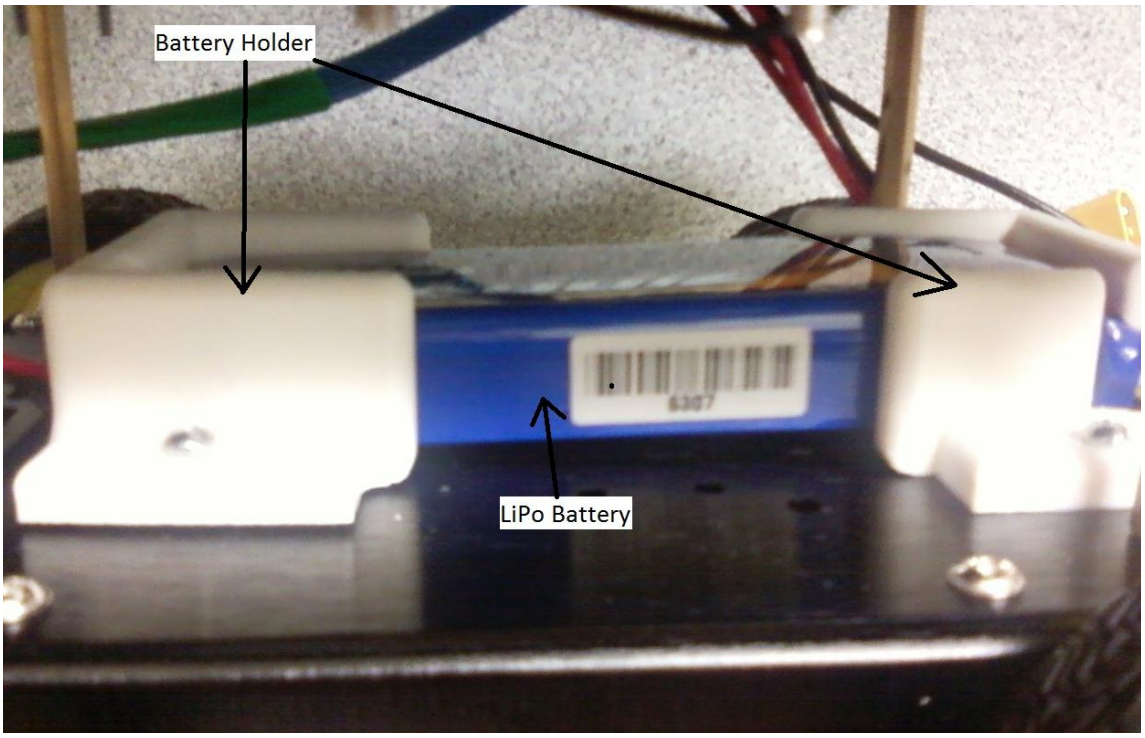

Fig. 4 Rapid Prototype of the Battery Holder

VI. Actuation

DC Motors

Two ServoCity DC micro gear motors are used to provide the robot with basic mobility. They are 12 volt motors with a max speed of 130 rpm. The have a stall torque of 50 oz-in and a stall current of 1600 mA. The robot is driven by the front two wheels while the back wheels are free spinning. Steering is achieved using differential drive between the front right and front left wheels.

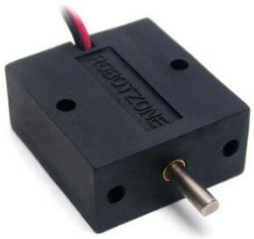

Fig. 5 Micro Gear DC Motor

Personal Emergency Response System

For this robot, the laptop computer will perform the same function as a PERS. The laptop features 4 gigabytes of RAM and runs a 64-bit version of Windows 7. When the robot discovers a human, the laptop will be able to connect to the internet and text a preprogrammed emergency contact number. In an earlier design, using a commercial PERS was considered. The robot would

carry the radio transmitter next to the Epiphany DIY as well as a recordable sound module. There were a couple reasons for moving away from this type actuation. The first issue with taking this route is that commercially available PERS are expensive and would require a lot of additional hardware such as a phone and a PERS console. The second issue is that it would be hard to test and demonstrate that the PERS works without actually calling emergency services.

VII. Sensors

Ultrasonic Range Finder

An ultrasonic range finder is mounted in the front of the robot. It is a Maxbotix SEN-00639 and operates between 2.5 and 5.5 volts. It consumes 2 mA, reads at a rate of 20 Hz, and features both analog and PWM output. For this project, only the analog output was used and provides a resolution of 10 mV/in. It has a range of 0 to 255 inches with little to no dead zone. It is used in conjunction with the infrared proximity sensors for basic obstacle avoidance. Characterization of the sensor is provided in the graph below. The experiment for how this characterization graph was derived is discussed in detail in the "Experimental Layout and Results" section of this report.

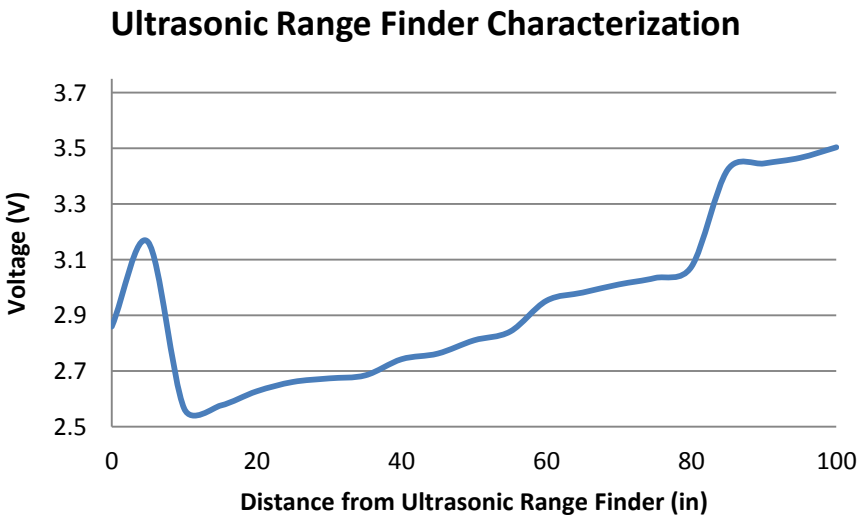

Fig. 6 Characterization of Ultrasonic Range Finder

Fig. 7 Ultrasonic Range Finder

Long Range Infrared Proximity Sensor

Two Sharp long range infrared GP2Y0A02YK0F sensors are mounted on the top level of the robot platform. One is angled at the front left corner of the robot, while the other is angled at the front right corner. The sensors have a range of 20 cm to 150 cm, consume 33 mA, and require an input voltage in the range of 4.5 to 5.5 VDC. The sensors output an analog voltage that varies between .4 V (150 cm) to 2.8 V (15 cm). They will be utilized along with the ultrasonic range finder for obstacle avoidance. Characterization of the two sensors is provided in the graphs below. The experiment for how these characterization graphs were derived is discussed in detail in the "Experimental Layout and Results" section of this report.

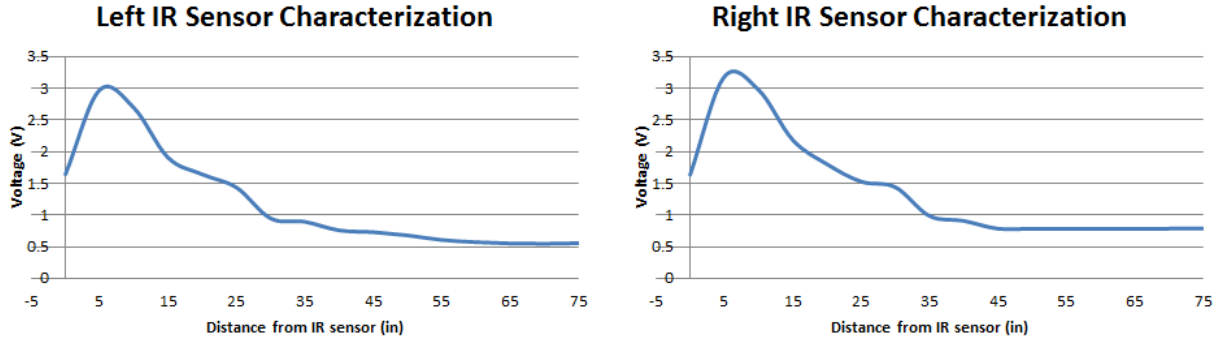

Fig. 8 Characterization of Left and Right IR Sensors

Fig. 9 Sharp Long Range Infrared Proximity Sensor

IP Camera

The IP Camera used for this robot is a Cisco Wireless N Internet Camera WVC80N IP camera. It communicates with a laptop computer using a Linksys wireless router. The camera is used in conjunction with OpenCV to detect if a human has fallen on the floor. The experiments for how I learned how to use this camera to detect faces is described in detail in the "Experimental Layout and Results" section of this report.

Fig. 10 Cisco Linksys IP Camera

Battery Monitor

Whenever using a LiPo battery, it is important to monitor the battery voltage so that it does not drop too low. This particular battery monitor is designed for a three cell LiPo battery which is used to power the robot. The characterization of the battery monitor is summarized in the table below.

Blue light shines	Blue light flashes	Red light shines	Red light flashes and beeps
11V and up	11.1V ~ 10.2V	10.2V ~ 9.9V	Below 9.9V

Fig. 11 Battery Monitor Characterization

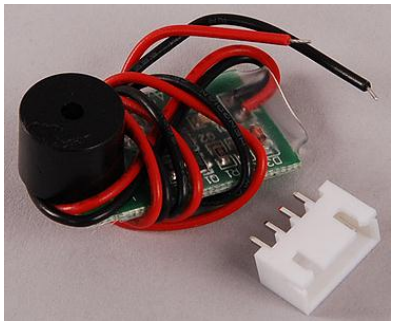

Fig. 12 3S Battery Monitor

Micro Switch

A digital micro switch is used to turn the motors off. It is very helpful to have, especially when troubleshooting a new program. It is much easier to disconnect power from the robot when you don't have to worry about it driving away while you pull the battery out. The switch used on the robot is in the normally closed configuration. The program that I used to test the switch is in the code labeled, "Limit Switch and Sensor Reading Program" which is located in the "Atmel Studio 6.0 Code" section of the Appendix. The switch was not used in the final version of the robot. It was replaced instead with a switch that physically kills power to the robot directly, rather than turns off the motors in the code. However, since the switch was used heavily in the experimentally phases of the design, it was included in this report for completeness.

Fig. 13 Micro Switch

VIII. Behaviors

Search Mode and Obstacle Avoidance

Once the robot is activated, it will begin navigating the home looking for a fallen human. Until it discovers a human, it will engage in a search behavior. While searching for a human, the robot will use its infrared and ultrasonic sensors to do basic obstacle avoidance. If the ultrasonic sensor senses an object directly in front of it, it will back up for a random amount of time and

then turn in a random direction using differential steering. If one of the IR sensors detects an object is within a certain threshold of the robot, it will reverse the direction of the opposite side's motor to cause the robot to veer away from the object. If both IR sensors detect an object is within a certain threshold, it will back up for a random amount of time and then turn in a random direction.

Emergency Response

If the IP camera and the OpenCV program detect a human lying on the floor, the laptop will send a signal to the robot via a Bluetooth connection. Once the robot receives confirmation that it has found someone, it will stop moving and activate the personal emergency response system (PERS). The laptop computer will be able to send a prewritten e-mail with a photo attachment along with a text message to a family member or neighbor as a means to obtain assistance. The recipients of the message will be able to view the attached photo to verify whether the emergency is authentic.

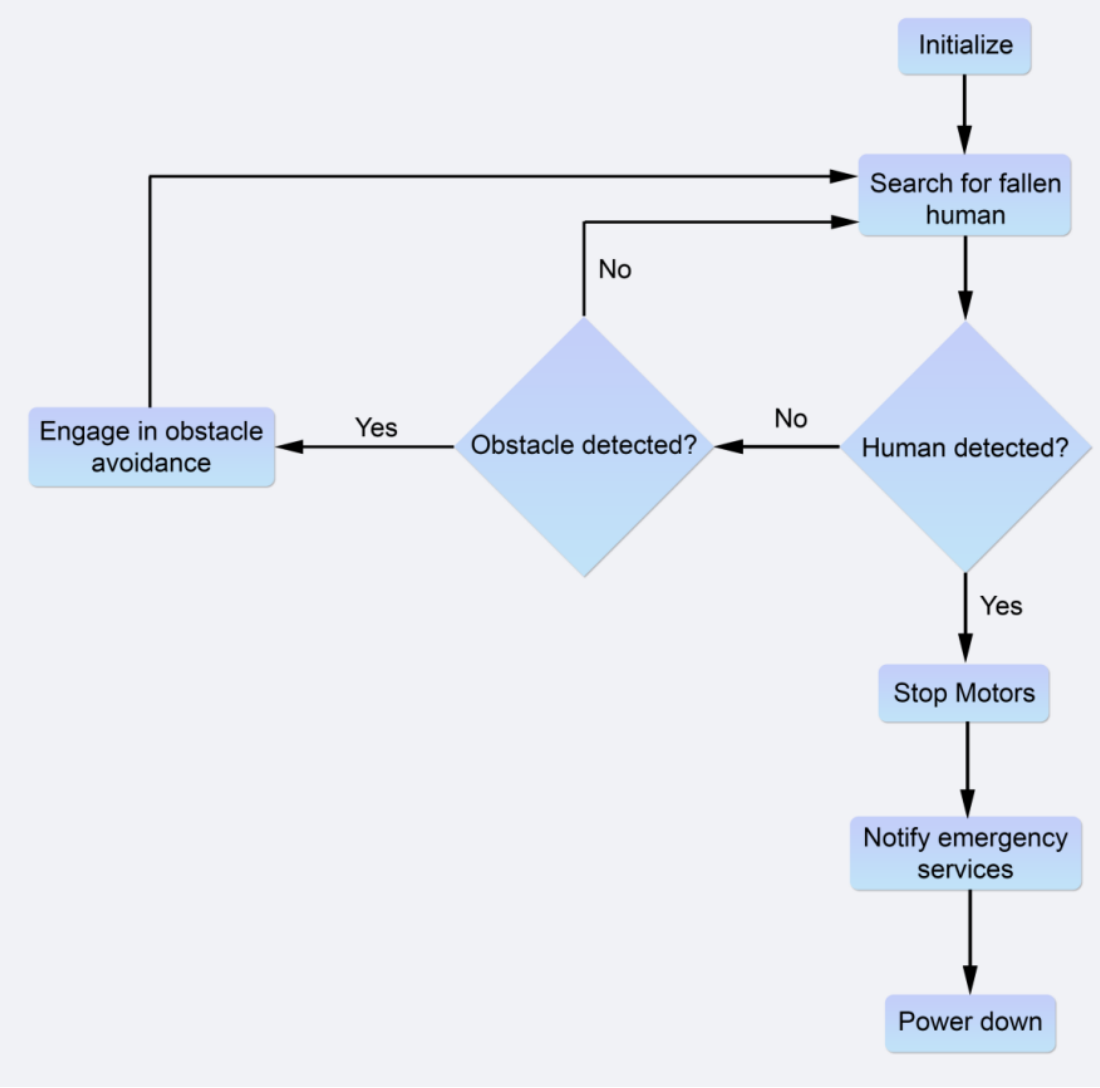

Fig. 14 Behavioral block diagram of the robot's operation

IX. Experimental Layout and Results

The two IR sensors were characterized by placing a wall made out of cardboard at a determined distance from them and reading the ADC value via a USB cable connecting the Epiphany DIY with the laptop. The software used to display the data coming into the computer was the X-CTU terminal. The same process was also used to characterize the ultrasonic range sensor. The results of these experiments can be found in the "Sensors" section of this report. The program that I used in these experiments is the code labeled, “Limit Switch and Sensor Reading Program” which is located in the “Atmel Studio 6.0 Code” section of the Appendix.

Once the sensors were characterized, I began testing my obstacle avoidance code. The code takes readings from the IR and ultrasonic sensors, and uses the data to avoid bumping into obstacles. To start my obstacle avoidance experiments, I tried only using two IR sensors. One faced the front-left corner of the robot, while the other faced the front right corner. When the left sensor detected an object close to it, the robot would turn to the right. If the right sensor detected an object nearby, it would turn the other way. For the most part this seemed to work well, but there were two issues that had to be addressed. First, when both sensors detected an object close to it, it would begin to wiggle back and forth and get stuck. This problem was fixed by adding a conditional statement for when both sensors detect a nearby object. If this case was true, then the robot was programmed to back up for a random amount of time and turn in a random direction. The second issue was caused by the IR sensors not facing directly the front of the robot, creating a blind spot. If the robot encountered a narrow obstacle (like a pole) directly in front of it, it would not be able to detect it and would ultimately crash into it. This was corrected by utilizing the ultrasonic sensor and mounting it directly in front of the robot. A calibration process for the IR sensors was added to the final version of the obstacle avoidance program and can be found in the “Atmel Studio 6.0 Code” section of the Appendix under the label of “Fallen Angel V_0.”

The next experiment I ran was on characterizing the IP camera, which is the special sensor in this project. Since controlling an e-mail account with a camera is not a trivial task, I broke the problem into smaller parts. With each success I added to the complexity of the code. For example, the first test experiment I ran was getting the laptop’s web camera to communicate with OpenCV. The next experiment involved getting the web camera to detect a face using OpenCV. Since I was new to face detection using Haar-like features, I started out with sample code that I found on the internet. [6] The sample code took some adjusting to get it to function properly on my laptop, but eventually it worked.

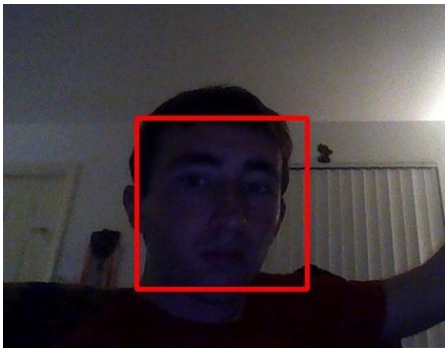

Fig. 15 Sample frame from the face detection OpenCV program

The next two issues I faced with the code were that it wouldn't run in real time nor was it able to get information from the IP camera. The sample code simply loaded an image, searched for faces, and then outputted another image with the faces highlighted if it found any. I then expanded on the code and ran an experiment to test whether it could load images in a loop, simulating real time usage. Once I had accomplished this, I learned how to save frames from a web camera into image files. Now I was able to load the frame that I had saved from my web camera and have the program detect faces in real time. The next step was transferring from the web camera as the image source to the IP camera. It turns out that this was as simple as changing one line of code in my program.

The last major issue I encountered while working on the face detection program was its sensitivity to the orientation of the face. The program would only be able to find a face if it was oriented in the upright position. In order to compensate for this, I rotated the raw feed from the camera both 90 degrees clockwise and counterclockwise. I then ran these images through the face detection program. This allows the program to find faces in three orientations, upright and rotated 90 degrees clockwise and counterclockwise. The last two are particularly useful since when someone falls on the ground they are more likely to have their head oriented in one of those positions rather than being upright.

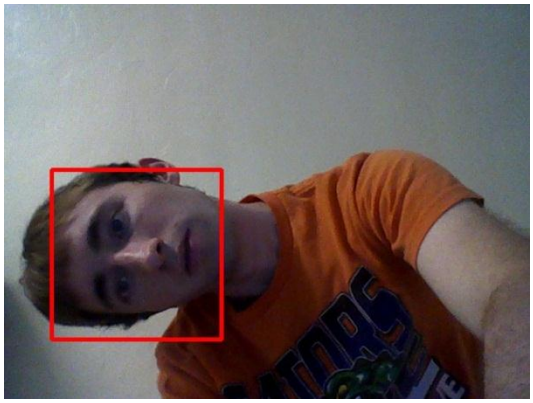

Fig. 16 Sample frame from the face detection OpenCV program

When using the IP camera, there seems to be a minute delay between when I start the program to when it is able to run properly. Otherwise it does exactly what it needs to do, detect faces from the IP camera in real time. The code that I used for these experiments is labeled "Face Detector v3.0" and is located in the "OpenCV Code" section of the Appendix.

Once I could detect faces reliably with the IP camera, the next step was getting the OpenCV code to call for help by sending e-mails and text messages. I did a lot of research in the subject and learned the basics of short message transfer protocol (SMTP). I found a video that demonstrated how easy it was to construct a Visual Basic application that could send prewritten e-mail messages with attachments automatically. [5] Once I had this working, I did some more research on how I could extend this program to also send text messages. I learned that each phone number has an e-mail address associated with it. When an e-mail is sent to this address, the phone treats it as a text message. Once I learned how to find the associated e-mail address for a given phone number, it was trivial to rewrite the program so that it could send text messages as well. As a bonus feature, I also learned how to add text-to-speech to application from another

video online. [4] It provides an audial cue to the victim when a call for help has been sent.

Now that the Visual Basic application was able to send e-mails and text messages, I still had the problem of how OpenCV was going to call that application when it found a face. By doing some research, I learned about the CMD in Windows and how system functions work in C++. This provided me with the tools to write a code that will run the Visual Basic application whenever a face is detected.

X. Future Work

In the "Experimental Layout and Results" section of the paper, I discuss how one of the disadvantages of using face detection as a means for the robot to detect a fallen human is that it is sensitive to the orientation of the person's face. What happens if the individual falls and lands on their back? There would be no way for the robot to detect that they have fallen. One way to resolve this issue is to have the robot instead detect for body profiles of people who have fallen and are lying on the ground. The reason that this was not implemented during this semester was due to time constraints. Building such a library is a very time consuming process. Creating a new library for people lying on the floor would require thousands of images, with a wide variety of camera angles, clothing variation, lighting conditions, body positions, etc. Then once the photos are taken, thousands of more photos would have to be taken that do not include the object. Ultimately, to make this project the best it can be, such a library would have to be created.

When things in the household are normal and there is no emergency, there is no reason for the robot to be active. However, how does the robot know when to start searching the house for someone who has fallen? One way to do this is to use a loud sound as a trigger to activate the robot. This is possible by using a microphone to monitor the sound level in the home. However, doing this leads to another problem. How can the robot differentiate between thunder and someone yelling and falling down on the floor? Another possible solution is to just have the robot make periodic searches of the house. Or perhaps an even better solution would be to integrate the Fallen Angel to a vacuum cleaning robot, so it clean the house as it searched for fallen humans. It would be worthwhile in the future to investigate which method of activation is best.

One final improvement that could be made to the robot is the ability to view the feed from the robot's IP camera remotely. A link could be provided in the preprogrammed e-mail so that the user could view the victim in real time to get a better grasp of the situation. This is possible by obtaining a DNS host, but it would slightly increase the overall cost of the robot.

XI. Conclusion

In conclusion, the Fallen Angel presents a novel solution to a very difficult problem. The Fallen Angel provides several advantages to currently available commercial personal emergency response systems. The first advantage is that it does not require the user to carry around a radio transmitter which can be invasive and uncomfortable. Commercial PERS also require that the user is capable of pressing the button on the transmitter. However, what if the user is suffering

from a condition that makes them unable to press the button, such as experiencing a seizure, heart attack, stroke, etc.? This leads to the second advantage of the Fallen Angel over commercial PERS. It can detect if someone has fallen, even if they are no longer conscious. The third advantage is that the user can decide who they want to receive the emergency texts and e-mails. This is a nice added feature since the user can add their neighbors to the robot's contact list, allowing for a potentially faster response time. A comparison table of typical response times of the Fallen Angel with commercial PERS can be found in the appendix. The final advantage of the Fallen Angel is that it has the potential to be more cost effective than the other commercial PERS. While the initial hardware investment is greater for the Fallen Angel, the lack of a monthly fee makes it one of the cheapest PERS alternatives within just one year of ownership. A lifetime cost comparison of the Fallen Angel compared to other commercial PERS can be found in the Appendix. Producing these robots in large scale would only further drive down the cost. Due to above reasons, I believe that the Fallen Angel would be a competitive alternative in the personal emergency response system market. Even better, it can be used in conjunction with a traditional PERS in order to ensure the greatest chance for a successful rescue.

XII. Documentation

[1] Centers for Disease Control and Prevention Accessed on 12 September 2012. http://www.cdc.gov/homeandrecreationalsafety/falls/adultfalls.html

[2] ENDependence Center of Northern Virginia, Inc. December 2009. Accessed on 12 September 2012. http://www.ecnv.org/FAQs/pers.html

[3] USLegal.com. Accessed on 12 September 2012. http://definitions.uslegal.com/p/personal-emergency-response-system/

[4] Amant, Timmy. "Tutorial 1: Microsoft Visual Studio 2010 - text to speech." 24 September 2010. Online video clip. YouTube. Accessed on 29 September 2012. http://www.youtube.com/watch?v=wxBjWbvbfoc

[5] TeamNiBiCnet. " Visual basic 2008/2010 Tutorial - Email Sender using Smtp" 28 August 2010. Online video clip. YouTube. Accessed on 23 September 2012. http://www.youtube.com/watch?v=9VBO5A-dNmo

[6] OpenCVWiki. SSteve. 25 August 2011. Accessed on 16 September 2012. http://opencv.willowgarage.com/wiki/FaceDetection

[7] Federal Trade Commission. Accessed on 12 September 2012. http://www.ftc.gov/bcp/edu/microsites/whocares/emergency.shtm

[8] Paul Viola & Michael Jones. "Robust Real-time Object Detection." International Journal of Computer Vision. Vancouver, Canada, JULY 13, 2001: pages 0 - 25.

[9] Medical Alert Advice. Accessed on 28 November 2012. http://www.medicalalertadvice.com/review-summary.php

XIII. Appendix

CAD Model Images

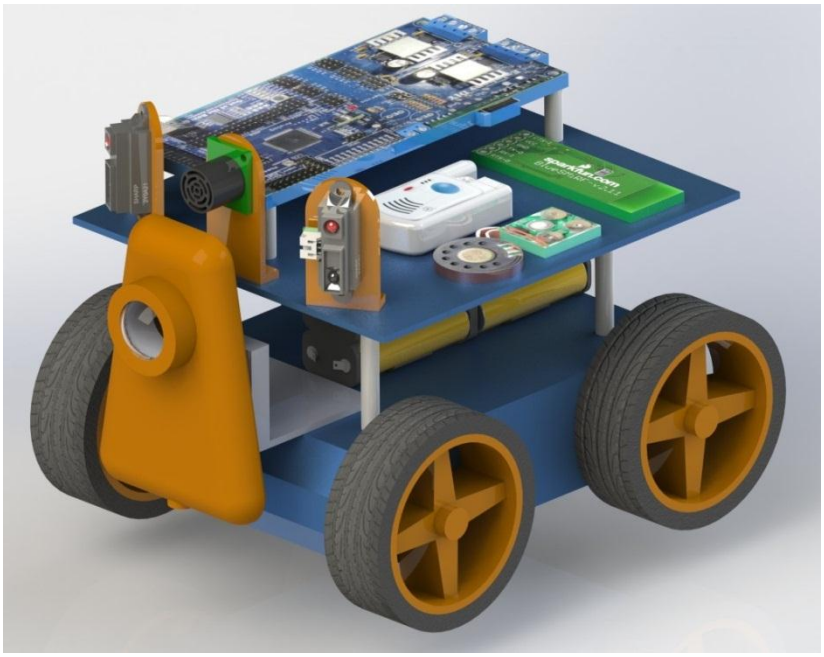

Fig. 17 Isometric view of the CAD model of the fully assembled robot

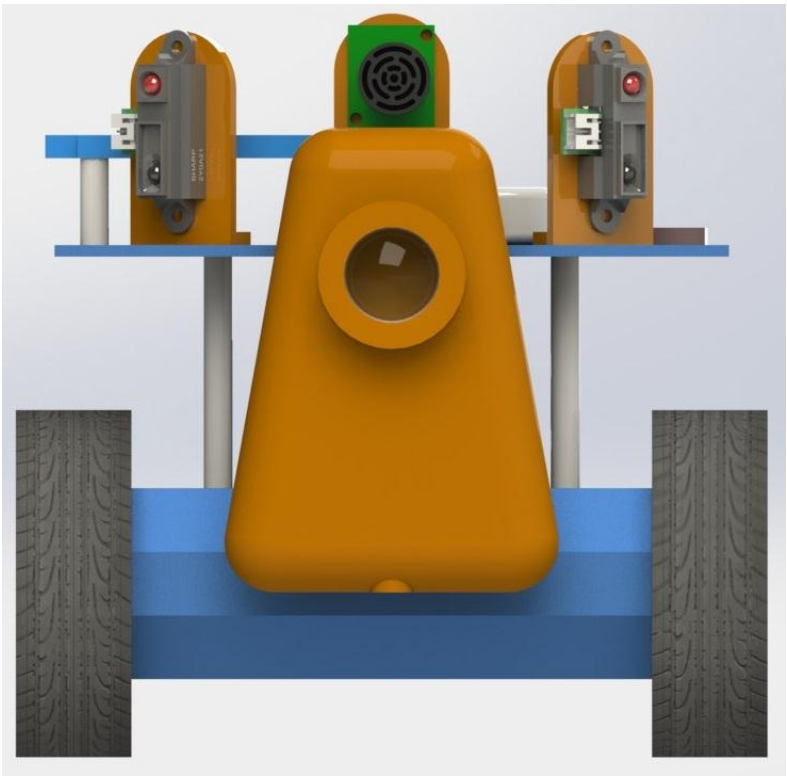

Fig. 18 Front view of the CAD model of the fully assembled robot

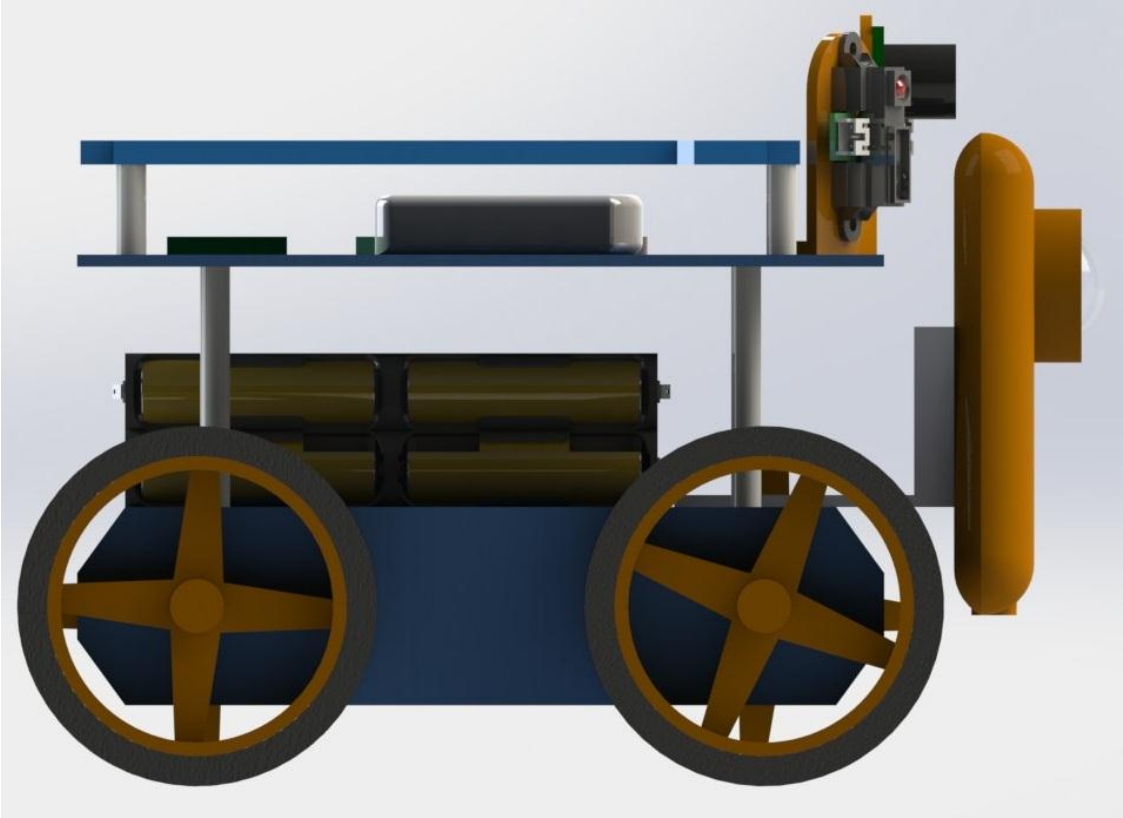

Fig. 19 Side view of the CAD model of the fully assembled robot

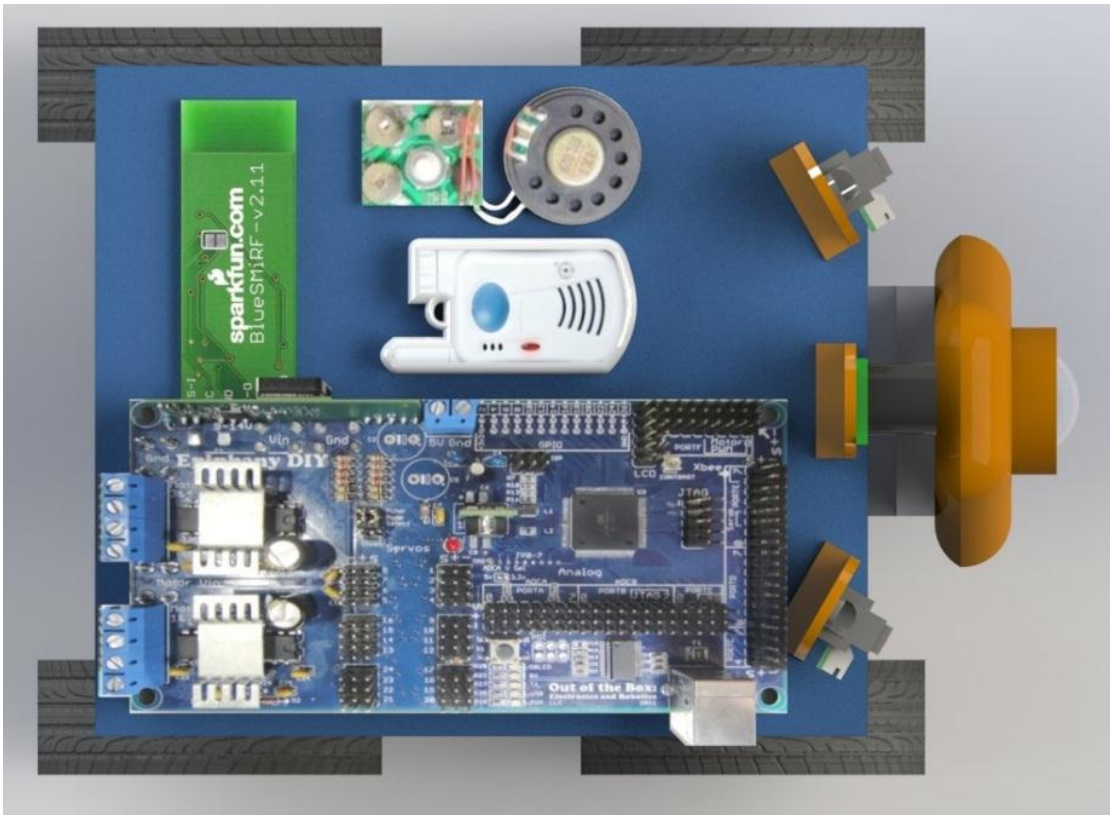

Fig. 20 Top view of the CAD model of the fully assembled robot

Average Response Time Comparison

Response time is defined as the time between when the user signals for help and when someone on the outside world is made aware of the user's call. For the commercial PERS, it is the amount of time between pressing the button on the transmitter and when someone answers the call at a monitoring station. In the Fallen Angel's case, it typically takes 5 to 10 seconds after when a face is detected before the predetermined contacts receive their e-mails and text messages. This assumes that emergency contact checks their phone or e-mail immediately after it is sent. Since there is no guarantee of this, I state that there is a potential for a faster response time.

PERS Provider	Average Response Time
Life Alert	30-50 seconds
Phillips Lifeline	20-30 seconds
ADT Companion	45 seconds
Fallen Angel	5 -10 seconds

Fig. 21Average Response Time Comparison

Cost Breakdown

Item	Cost
Epiphany Board	$115.00
Servo +	$40.00
Motor Driver Duo	$30.00
Sensors	$36.88
Ultra Sonic Sensor	$0.00
IP Camera	$111.64
Chassis	$70.00
Motors	$36.97
Wheel Hubs	$20.97
Blue SMiRF	$43.59
LiPo Battery	$13.82
Battery Monitor	$3.99
Total:	**$522.86**

Fig. 22Cost Breakdown of the Fallen Angel

Lifetime Cost Comparison

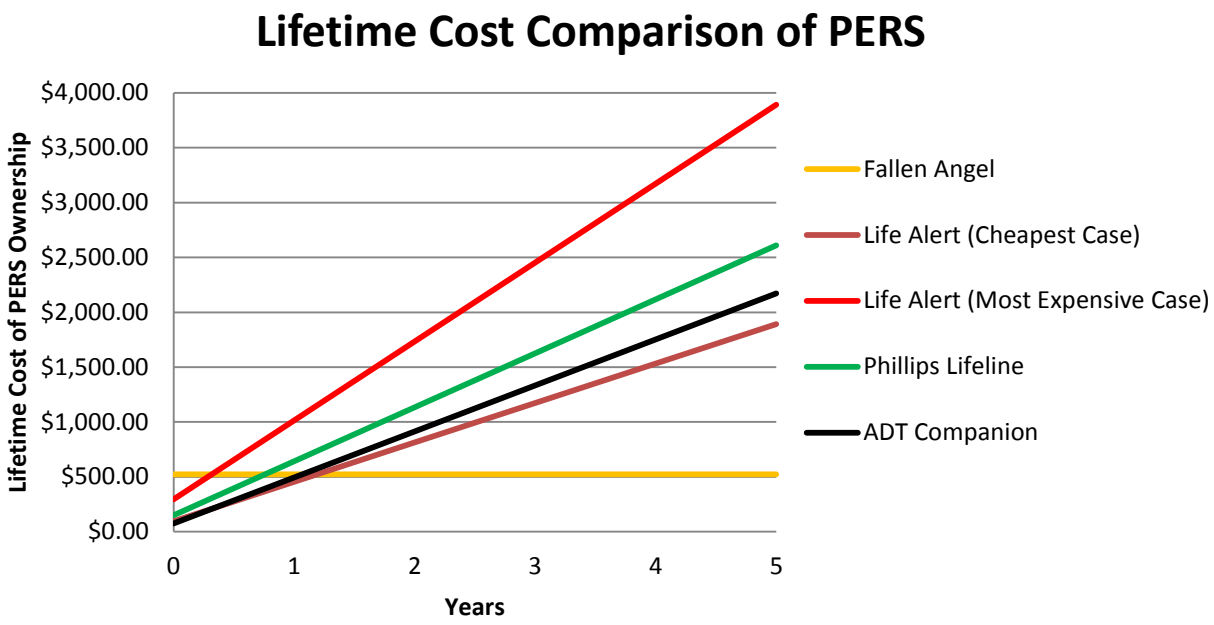

Fig. 23 Lifetime Cost Comparison of PERS

Visual Basic Code

```
/*****************************************************************
 * Personal Emergency Response System
 * Created: 09/23/2012 7:33 PM
 *  Author: Daniel Frank
 *
 * Description: This program sends a prewritten message to my g-mail account
 *                    as well as send a text to my cell phone. It also uses text-to-
 *                    speech to give an audio cue that help has been sent for
 *****************************************************************/
Imports System.Net.Mail
Namespace My

    Partial Friend Class MyApplication
    Dim message As New MailMessage
    Dim smtp As New SmtpClient intialize SMTP
    Dim SAPI ‘initializes text-to-speech

Private Sub MyApplication_Startup(ByVal sender As Object, ByVal e As
Microsoft.VisualBasic.ApplicationServices.StartupEventArgs) Handles Me.Startup
       message.From = New MailAddress("dzf209@gmail.com") ‘define sender address
            message.To.Add("dzf209@gmail.com") ‘address to send e-mail to
            message.To.Add("9085284218@txt.att.net") ‘phone number to send text to
       message.Body = "This is a test of an emergency notification system" ‘message text
       message.Subject = "This is only a test" ‘message subject
       message.Attachments.Add(New Attachment("face_found.jpg")) ‘attach a photo of the fallen person
       message.Priority = MailPriority.Normal ‘priority of the message

            'SMTP Client Settings'
       smtp.EnableSsl = True
       smtp.Port = "587"
       smtp.Host = "smtp.gmail.com"
       smtp.Credentials = New Net.NetworkCredential("dzf209@gmail.com", "*****")
       smtp.Send(message)

           SAPI = CreateObject("sapi.spvoice")
                'Text is read aloud to notify the victim that help is on its way
           SAPI.Speak("Emergency services were notified. Help is on the way")

                MsgBox("Message was sent!") ‘visual cue that message was sent
        End Sub

Private Sub MyApplication_StartupNextInstance(ByVal sender As Object, ByVal e As
Microsoft.VisualBasic.ApplicationServices.StartupNextInstanceEventArgs) Handles Me.StartupNextInstance

End Sub
End Class

End Namespace
```

Atmel Studio 6.0 Code

```
/*************************************************************
 * Analog Sensor Read
 * Created: 10/05/2012 9:53 AM
 *  Author: Daniel Frank
 *
 * Description: This program will read an analog sensor attached to pin 0 in
 *                    Port A and display it on the X-CTU terminal
 *************************************************************/
#include <avr/io.h>
#include <util/delay.h>
#include <math.h>
#include "clock.h"
#include "uart.h"
#include "adc.h"
#include <stdio.h>

int main(void)
{
        //Initialize phase
        clockInit();
        motorInit();
        adcInit(&ADCA);
        usartInit(&USARTC0,115200);
        sei();

while(1)
   {
        fprintf(&USB_str,"%d \r\n", analogRead(&ADCA, 0));  //prints sensor reading to X-CTU
                                                            //Terminal
        _delay_ms(100);  //delay makes it easier to read the sensor data in real-time
    }
}
```

```
/*****************************************************************
 * Motor Test
 * Created: 10/05/2012 5:14 PM
 *  Author: Daniel Frank
 *
 * Description: This program will test if the motors are working
 *****************************************************************/

#include <avr/io.h>
#include <util/delay.h>
#include <math.h>
#include "clock.h"
#include "uart.h"
#include "motor.h"
#include "adc.h"
#include <stdio.h>

int main(void)
{
	//Initialize phase
	clockInit();
	motorInit();
	adcInit(&ADCA);
	usartInit(&USARTC0,115200);
	sei();

while(1)
   {
	setMotorEffort(1,800, MOTOR_DIR_FORWARD); //Drive Left Motor Forward
	setMotorEffort(2,800, MOTOR_DIR_FORWARD);  //Drive Right Motor Forward
   }
}
```

```
/*****************************************************************
 * Motor and Sensor Test
 * Created: 10/05/2012 6:40 PM
 *  Author: Daniel Frank
 *
 * Description: This program will test if the motors are able to be controlled by
 *                  an IR sensor. Sensor readings will displayed on the X-CTU
 *                  terminal
 *****************************************************************/

#include <avr/io.h>
#include <util/delay.h>
#include <math.h>
#include "clock.h"
#include "ATtinyServo.h"
#include "uart.h"
#include "adc.h"
#include "motor.h"
#include <stdio.h>

int main(void)
{
        //Initialize phase
        clockInit();
        motorInit();
        adcInit(&ADCA);
        ATtinyServoInit();
        usartInit(&USARTC0,115200);
        sei();

        while(1)
   {
        //Display sensor data to the X-CTU terminal
        fprintf(&USB_str,"%d \t\t", analogRead(&ADCA, 0)); //Left IR Sensor
        fprintf(&USB_str,"%d \r\n", analogRead(&ADCA, 2)); // Right IR Sensor

        if (analogRead(&ADCA, 2) < 2000){
                setMotorEffort(1,800, MOTOR_DIR_FORWARD); // Drive Left Motor Forward
                setMotorEffort(2,800, MOTOR_DIR_FORWARD);  //Drive Right Motor Forward
        }

        else {
                 setMotorEffort(1,0, MOTOR_DIR_FORWARD); //Stop Left Motor
                 setMotorEffort(2,0, MOTOR_DIR_BACKWARD); //Stop Right Motor
        }

        _delay_ms(100);
        }
}
```

```
/*****************************************************************
 * Obstacle Avoidance 1
 * Created: 10/05/2012 7:12 PM
 * Author: Daniel Frank
 *
 * Description: This is an obstacle avoidance program. When one
 *              of the IR sensors reads a value within a
 *              predefined threshold value, it will reverse the
 *              motor direction while keeping the same speed
 *****************************************************************/
#include <avr/io.h>
#include <util/delay.h>
#include <math.h>
#include "clock.h"
#include "ATtinyServo.h"
#include "uart.h"
#include "adc.h"
#include "motor.h"
#include <stdio.h>

int main(void){
        //Initialize phase
        clockInit();
        motorInit();
        adcInit(&ADCA);
        ATtinyServoInit();
        usartInit(&USARTC0,115200);
        sei();

        int IR_Left, IR_Right; //define IR sensor variables

        while(1)
        {
        IR_Left = analogRead(&ADCA, 0); //Read Left IR sensor
        IR_Right = analogRead(&ADCA, 2); //Read Right IR sensor

        if(IR_Right < 2000){
                setMotorEffort(1,800, MOTOR_DIR_FORWARD); // Drive Left Motor Forward
        }
        else{
                setMotorEffort(1,800, MOTOR_DIR_BACKWARD); // Drive Left Motor Backward
        }
        if(IR_Left < 2000){
                setMotorEffort(2,800, MOTOR_DIR_FORWARD); // Drive Right Motor Forward
        }
        else{
                setMotorEffort(2,800, MOTOR_DIR_BACKWARD); // Drive Right Motor Backward
        }
        _delay_ms(100);
        }
}
```

```
/*****************************************************************
 * Obstacle Avoidance 2
 * Created: 10/05/2012 7:57 PM
 * Author: Daniel Frank
 *
 * Description: This is an obstacle avoidance program. The
 *                  motors are kept in the forward position and the
 *                  the magnitude of their speed is a function of IR
 *                  sensor readings
 *****************************************************************/
#include <avr/io.h>
#include <util/delay.h>
#include <math.h>
#include "clock.h"
#include "ATtinyServo.h"
#include "uart.h"
#include "adc.h"
#include "motor.h"
#include <stdio.h>

int main(void)
{
        //Initialize phase
        clockInit();
        motorInit();
        adcInit(&ADCA);
        ATtinyServoInit();
        usartInit(&USARTC0,115200);
        sei();

        //define variables
        int IR_Left_Min, IR_Right_Min, IR_Left, IR_Right, Motor_Left, Motor_Right;

        //Sensor parameters gathered from characterization experiment
        IR_Left_Min = 220;
        IR_Right_Min = 560;

        while(1)
  {
        IR_Left = analogRead(&ADCA, 0); //read Left IR sensor
        Motor_Left = 600 + (IR_Left - IR_Left_Min)/10; //generate left motor signal from
                                                        //sensor feedback

        IR_Right = analogRead(&ADCA, 2); //read Right IR sensor
        Motor_Right = 600 + (IR_Right - IR_Right_Min)/10; //generate right motor signal
                                                          //from sensor feedback
        if(Motor_Left > 1000){
                Motor_Left = 1000; //saturates left motor signal
        }

        if(Motor_Right > 1000){
```

```
                Motor_Right = 1000; //saturates right motor signal
        }

        Display sensor data to the X-CTU terminal
        fprintf(&USB_str,"%d \t\t", Motor_Left); //displays Left Motor Signal
        fprintf(&USB_str,"%d \r\n", Motor_Right); //displays Right Motor Signal

        _delay_ms(100);
        }
}
```

```
/*****************************************************************
 * Limit Switch and Sensor Reading Program
 * Created: 10/10/2012 8:31 PM
 *  Author: Daniel Frank
 *
 * Description: This program will test will output both IR
 *                  sensors, the ultrasonic sensor, and PortD which
 *                  is connected to a limit switch.
*****************************************************************/

#include <avr/io.h>
#include <util/delay.h>
#include <math.h>
#include "clock.h"
#include "ATtinyServo.h"
#include "uart.h"
#include "adc.h"
#include "motor.h"
#include <stdio.h>

int main(void)
{
        //Initialize phase
        clockInit();
        motorInit();
        adcInit(&ADCA);
        ATtinyServoInit();
        usartInit(&USARTC0,115200);
        sei();

        PORTD.DIRCLR = 0xFF; //set all of pins of port D to inputs
        PORTD.PIN1CTRL = 0x18; //set pin 1 to pull up

        while(1)
   {
        //Display sensor data to the X-CTU terminal
        fprintf(&USB_str,"%d \t\t", analogRead(&ADCA, 0)); //Left IR Sensor
        fprintf(&USB_str,"%d \t\t", analogRead(&ADCA, 2)); //Right IR Sensor
        fprintf(&USB_str,"%d \t\t", analogRead(&ADCA, 4)); // Ultra Sonic Sensor
        fprintf(&USB_str,"%d \r\n", PORTD.IN); // Port D

        _delay_ms(100);
        }
}
```

```
/*****************************************************************
 * OFF Switch
 * Created: 10/10/2012 8:53PM
 * Author: Daniel Frank
 *
 * Description: This program will use the digital switch in port
 *                  D to turn off the motors
*****************************************************************/

#include <avr/io.h>
#include <util/delay.h>
#include <math.h>
#include "clock.h"
#include "ATtinyServo.h"
#include "uart.h"
#include "adc.h"
#include "motor.h"
#include <stdio.h>

int main(void)
{
        //Initialize phase
        clockInit();
        motorInit();
        adcInit(&ADCA);
        ATtinyServoInit();
        usartInit(&USARTC0,115200);
        sei();

        PORTD.DIRCLR = 0xFF; //set all of pins of port D to inputs
        PORTD.PIN1CTRL = 0x18; //set pin 1 to pull up

        while(1){

        setMotorEffort(1,800, MOTOR_DIR_FORWARD); // Drive Left Motor Forward
        setMotorEffort(2,800, MOTOR_DIR_FORWARD);  //Drive Right Motor Forward

        if(PORTD.IN & 0xFF){
                while(1){
                        setMotorEffort(1,0, MOTOR_DIR_FORWARD); // Stop Left Motor Forward
                        setMotorEffort(2,0, MOTOR_DIR_FORWARD); // Stop Right Motor Forward
                }
        }

        _delay_ms(100);
        }
}
```

```
/*****************************************************************
 * Obstacle Avoidance 4
 * Created: 10/31/2012 9:12 PM
 * Author: Daniel Frank, adapted from code by Joshua Weaver
 *
 * Description: This is an obstacle avoidance program. When one
 *              of the IR sensors reads a value within a
 *              predefined threshold value, it will reverse the
 *              motor direction while keeping the same speed,
 *              also avoids getting stuck in corners
 *              The BlueSMiRF sections of the code were adapted
 *              from code provided by Joshua Weaver
 *****************************************************************/

#include <avr/io.h>
#include <util/delay.h>
#include <math.h>

#include "clock.h"
#include "ATtinyServo.h"
#include "uart.h"
#include "adc.h"
#include "motor.h"
#include <stdio.h>

// Defines
#define Blue_str usartD0_str

int main(void)
{
	// Initialization
	clockInit();
	adcInit(&ADCA);
	motorInit();
	ATtinyServoInit();
	usartInit(&USARTC0,115200); // Initialize USB_str port and interrupts
	usartInit(&USARTD0,115200); // Initialize blueSMiRF port and interrupts, pin2 RX, pin3 TX

	//PORTB.DIRCLR = 0xFF; //set all of pins of port B to inputs
	//PORTB.PIN1CTRL = 0x18; //set pin 1 to pull up

	// Setup ADC Mux Channels
	adcChannelMux(&ADCA,0,0);
	adcChannelMux(&ADCA,1,2);

	// Setup Port pin directions
	PORTC.DIRSET = 0x08;		// Set TX bit to output for USB_str
	PORTD.DIRSET = 0x08;		// Set TX bit to output for usartD0_str (blueSMiRF)
	PORTR.DIRSET = 0x02;		// Debug LED

	// Enable global interrupts.
```

```
	sei();

	// 5 Second Startup hold.
	for (int i = 5; i >= 1; i--) {
		fprintf(&USB_str, "Starting in %d...\r\n", i);

		PORTR.DIRTGL = 0x02;  // BLINK LED
		_delay_ms(500);

		PORTR.DIRTGL = 0x02;  // BLINK LED
		_delay_ms(500);
	}

	// Send message to connected devices as a checkup
	fprintf(&Blue_str, "BlueTooth Checkup.\r\n");


	// STARTING MAIN LOOP
	fprintf(&USB_str, "Starting main loop.\r\n");

	int IR_Left, IR_Right, Ultra_Sonic, random_delay; //define IR sensor variables

	while(1)
{
		// Repeat character entered from blueSMiRF to itself.
		// Note that this only received a single character at a time.
		// A While loop looking for an ending character can be used to
		// receive a string.
		if (dataInBufD0()) {
			char test[10];

			fscanf(&Blue_str,"%c",&test);

			// Here we check to see if a key has been pressed (1,2)
			if (strcmp(test, "1") == 0) {
				while(1){
					fprintf(&Blue_str, "Face was found.\r\n");
					setMotorEffort(1,0, MOTOR_DIR_FORWARD);
					setMotorEffort(2,0, MOTOR_DIR_FORWARD);
					}
			}
		}
		IR_Left = analogRead(&ADCA, 0); //Read Left IR sensor
		IR_Right = analogRead(&ADCA, 2); //Read Right IR sensor
		Ultra_Sonic = analogRead(&ADCA, 1); //Read Ultra Sonic sensor

		fprintf(&USB_str,"%d \t\t", analogRead(&ADCA, 0)); //Left IR Sensor
		fprintf(&USB_str,"%d \t\t", analogRead(&ADCA, 2)); //Right IR Sensor
		fprintf(&USB_str,"%d \r\n", analogRead(&ADCA, 4)); // Ultra Sonic Sensor
```

```
                if(IR_Right <1800){
                setMotorEffort(1,800, MOTOR_DIR_FORWARD); // Drive Left Motor Forward
                }
                else{
                setMotorEffort(1,800, MOTOR_DIR_BACKWARD); // Drive Left Motor Backward
                }
                if(IR_Left < 1800){
                setMotorEffort(2,800, MOTOR_DIR_FORWARD); // Drive Right Motor Forward
                }
                else{
                setMotorEffort(2,800, MOTOR_DIR_BACKWARD); // Drive Right Motor Backward
                }

                if(IR_Right > 1800 && IR_Left > 1800){
                        setMotorEffort(1,800, MOTOR_DIR_BACKWARD);
                        setMotorEffort(2,800, MOTOR_DIR_BACKWARD);
                        _delay_ms(1000);

                        if(Ultra_Sonic % 2 == 1){
                                setMotorEffort(1,800, MOTOR_DIR_FORWARD);
                                setMotorEffort(2,800, MOTOR_DIR_BACKWARD);
                                _delay_ms(1000);
                        }
                        else{
                                setMotorEffort(1,800, MOTOR_DIR_BACKWARD);
                                setMotorEffort(2,800, MOTOR_DIR_FORWARD);
                                _delay_ms(1000);
                        }
                }

                PORTR.DIRTGL = 0x02;  // HEARTBEAT LED
                _delay_ms(100);           // 100 ms loop
        }
}
```

```
/*****************************************************************************
 * Fallen Angel V_0
 * Created: 11/18/2012 1:54 PM
 * Author: Daniel Frank, sections of the code was adapted from code by Joshua Weaver
 *
 * Description: This is an obstacle avoidance and calibration
 *              program. When one of the IR or ultrasonic sensors
 *              reads a value within a threshold value defined by
 *              the calibration process, it will reverse the
 *              motor direction, it also avoids getting stuck in
 *              corners. The bluesmirf sections of the code were
 *              adapted from code provided by Joshua Weaver
 *****************************************************************************/
#include <avr/io.h>
#include <util/delay.h>
#include <math.h>
#include "clock.h"
#include "ATtinyServo.h"
#include "uart.h"
#include "adc.h"
#include "motor.h"
#include <stdio.h>

// Defines
#define Blue_str usartD0_str

int main(void)
{
        // Initialization
        clockInit();
        adcInit(&ADCA);
        adcInit(&ADCB);
        motorInit();
        ATtinyServoInit();
        usartInit(&USARTC0,115200); // Initialize USB_str port and interrupts
        usartInit(&USARTD0,115200); // Initialize blueSMiRF port and interrupts, pin2 RX,
                                    //pin3 TX

        // Setup ADC Mux Channels
        adcChannelMux(&ADCA,0,0);
        adcChannelMux(&ADCA,1,2);

        // Setup Port pin directions
        PORTC.DIRSET = 0x08;      // Set TX bit to output for USB_str
        PORTD.DIRSET = 0x08;      // Set TX bit to output for usartD0_str (blueSMiRF)
        PORTR.DIRSET = 0x02;      // Debug LED
```

```
	// Enable global interrupts.
	sei();

	int IR_Left, IR_Right, Ultra_Sonic, IR_Left_Threshold, IR_Right_Threshold,
	Ultra_Sonic_Threshold; //define IR sensor variables

	// 5 Second Startup hold.
	for (int i = 3; i >= 1; i--) {
		fprintf(&USB_str, "Starting in %d...\r\n", i);

		PORTR.DIRTGL = 0x02;  // BLINK LED
		_delay_ms(500);

		PORTR.DIRTGL = 0x02;  // BLINK LED
		_delay_ms(500);
	}
	IR_Left_Threshold = analogRead(&ADCA, 0); //Read Left IR sensor
	IR_Right_Threshold = analogRead(&ADCA, 2); //Read Right IR sensor
	Ultra_Sonic_Threshold = 250; //Read Ultra Sonic sensor

	// Send message to connected devices as a checkup
	fprintf(&Blue_str, "BlueTooth Checkup.\r\n");

	// STARTING MAIN LOOP
	fprintf(&USB_str, "Starting main loop.\r\n");

	while(1)
{
		// Repeat character entered from blueSMiRF to itself.
		// Note that this only received a single character at a time.
		// A While loop looking for an ending character can be used to
		// receive a string.
		if (dataInBufD0()) {
			char test[10];

			fscanf(&Blue_str,"%c",&test);

			// Here we check to see if a key has been pressed (1,2)
			if (strcmp(test, "1") == 0) {
				while(1){
					fprintf(&Blue_str, "Face was found.\r\n");
					setMotorEffort(1,0, MOTOR_DIR_FORWARD);
					setMotorEffort(2,0, MOTOR_DIR_FORWARD);
					}
			}
```

```
                }
                IR_Left = analogRead(&ADCA, 0); //Read Left IR sensor
                IR_Right = analogRead(&ADCA, 2); //Read Right IR sensor
                Ultra_Sonic = analogRead(&ADCB, 0); //Read Ultra Sonic sensor

                fprintf(&USB_str,"%d \t\t", analogRead(&ADCA, 0)); //Left IR Sensor
                fprintf(&USB_str,"%d \t\t", analogRead(&ADCA, 2)); //Right IR Sensor
                fprintf(&USB_str,"%d \r\n", analogRead(&ADCB, 0)); // Ultra Sonic Sensor

                if(IR_Right <IR_Right_Threshold){
                setMotorEffort(1,700, MOTOR_DIR_FORWARD); // Drive Left Motor Forward
                }
                else{
                setMotorEffort(1,1000, MOTOR_DIR_BACKWARD); // Drive Left Motor
                                                                //Backward
                }
                if(IR_Left < IR_Left_Threshold){
                setMotorEffort(2,700, MOTOR_DIR_FORWARD); // Drive Right Motor
                                                            //Forward
                }
                else{
                setMotorEffort(2,1000, MOTOR_DIR_BACKWARD); // Drive Right Motor
                                                                //Backward
                }

                if(IR_Right > IR_Right_Threshold && IR_Left > IR_Left_Threshold ||
                Ultra_Sonic <Ultra_Sonic_Threshold){
                        setMotorEffort(1,700, MOTOR_DIR_BACKWARD);
                        setMotorEffort(2,700, MOTOR_DIR_BACKWARD);
                        _delay_ms(1000);

                        if(Ultra_Sonic % 2 == 1){
                                setMotorEffort(1,1000, MOTOR_DIR_FORWARD);
                                setMotorEffort(2,1000, MOTOR_DIR_BACKWARD);
                                _delay_ms(1000);
                        }
                        else{
                                setMotorEffort(1,1000, MOTOR_DIR_BACKWARD);
                                setMotorEffort(2,1000, MOTOR_DIR_FORWARD);
                                _delay_ms(1000);
                        }
                }
                PORTR.DIRTGL = 0x02;  // HEARTBEAT LED
                _delay_ms(100);         // 100 ms loop
        }
}
```

```
/*********************************************************************************
* Face Detector v3.0
* Created: 11/025/2012 12:27 PM
* Author: Daniel Frank
*
* Description: This program will uses Haar-like features to detect faces using
*              the IP camera, once a face is detected it signals to the robot
*              that it has found a face and activates the PERS
*********************************************************************************/

#include <cxcore.h>
#include <highgui.h>
#include <cv.h>
#include <ml.h>
#include <iostream>
#include "stdafx.h"
#include "stdafx.cpp"
#include "targetver.h"
#include <Windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <math.h>
#include <float.h>
#include <limits.h>
#include <time.h>
#include <ctype.h>

using namespace std;
int count_now = 0, count_1 = 0, count_2 = 0, countr_now = 0, countr_1 = 0, countr_2 = 0,
countl_now = 0, countl_1 = 0, countl_2 = 0, found=0;

// Create memory for calculations
static CvMemStorage* storage = 0;

// Create a new Haar classifier
static CvHaarClassifierCascade* cascade = 0;

// Create a string that contains the exact cascade name
const char* cascade_name = "C:/opencV/data/haarcascades/haarcascade_frontalface_alt.xml";

// Function prototype for detecting and drawing an object from an image
void detect_and_draw( IplImage* image );

// Main function, defines the entry point for the program.
int _tmain(int argc, _TCHAR* argv[])
{
       //CvCapture* capture =  cvCreateCameraCapture(1);
       CvCapture* capture = cvCreateFileCapture("http://192.168.1.101/img/video.mjpeg");

       IplImage* frame = NULL;

        while ( found == 0 )
    {
        // Get one frame
```

```
        IplImage* frame = cvQueryFrame( capture );

        if ( !frame )
        {
            fprintf( stderr, "ERROR: frame is null...\n" );
        }

        detect_and_draw(frame);
        if ( (cvWaitKey(10) & 255) == 27 ) break;
    }
    // Release the capture device housekeeping
    cvReleaseCapture( &capture );

    if (found != 0) {
           return 0; //to indicate successfull execution of the program
       }
}
void detect_and_draw( IplImage* img )
{
    // Create memory for calculations
    static CvMemStorage* storage = 0;
    // Create a new Haar classifier
    static CvHaarClassifierCascade* cascade = 0;

    int scale = 1, face_detect1=0, face_detect2=0, face_detect3=0;
    // Create a new image based on the input image
    IplImage* temp = cvCreateImage( cvSize(img->width/scale,img->height/scale), 8, 3 );

       IplImage *transposeImage1 = cvCreateImage( cvSize(img->height/scale, img-
>width/scale), 8, 3 );
       IplImage *transposeImage2 = cvCreateImage( cvSize(img->height/scale, img-
>width/scale), 8, 3 );

       cvTranspose (img, transposeImage1);
       cvFlip(transposeImage1,transposeImage2,0);

    // Create two points to represent the face locations
    CvPoint pt1, pt2;
    int i;
    // Load the HaarClassifierCascade
    cascade = (CvHaarClassifierCascade*)cvLoad( cascade_name, 0, 0, 0 );

    // Check whether the cascade has loaded successfully. Else report and error and quit
    if( !cascade )
    {
        fprintf( stderr, "ERROR: Could not load classifier cascade\n" );
        return;
    }
    // Allocate the memory storage
    storage = cvCreateMemStorage(0);
    // Create a new named window with title: result
    cvNamedWindow( "Fallen Angel", 1 );
       //cvNamedWindow( "resulttranspose1", 1 );
       //cvNamedWindow( "resulttranspose2", 1 );

    // Clear the memory storage which was used before
    cvClearMemStorage( storage );
    // Find whether the cascade is loaded, to find the faces. If yes, then:
```

```
 if( cascade )
 {
//There can be more than one face in an image. So create a growable sequence of faces.
// Detect the objects and store them in the sequence

          CvSeq* faces1 = cvHaarDetectObjects( img, cascade, storage,
        1.1, 3, CV_HAAR_DO_CANNY_PRUNING,
        cvSize(40, 40) );

          face_detect1 = (faces1 ? faces1->total : 0);

    // Loop the number of faces found.
    for( i = 0; i < (faces1 ? faces1->total : 0); i++ )
    {   // Create a new rectangle for drawing the face
        CvRect* r = (CvRect*)cvGetSeqElem( faces1, i );
        // Find the dimensions of the face,and scale it if necessary
        pt1.x =r->x*scale;
        pt2.x =(r->x+r->width)*scale;
        pt1.y = r->y*scale;
        pt2.y = (r->y+r->height)*scale;
        // Draw the rectangle in the input image
        cvRectangle( img, pt1, pt2, CV_RGB(255,0,0), 3, 8, 0 );
        }
          CvSeq* faces2 = cvHaarDetectObjects( transposeImage1, cascade, storage,
        1.1, 3, CV_HAAR_DO_CANNY_PRUNING,
        cvSize(40, 40) );

          face_detect2 = (faces2 ? faces2->total : 0);

    // Loop the number of faces found.
    for( i = 0; i < (faces2 ? faces2->total : 0); i++ )
    {   // Create a new rectangle for drawing the face
        CvRect* r = (CvRect*)cvGetSeqElem( faces2, i );
        // Find the dimensions of the face,and scale it if necessary
        pt1.x = (r->y*scale);
        pt2.x = ((r->y+r->width)*scale);
        pt1.y = r->x*scale;
        pt2.y = (r->x+r->height)*scale;
        // Draw the rectangle in the input image
        cvRectangle( img, pt1, pt2, CV_RGB(255,0,0), 3, 8, 0 );
        }
          CvSeq* faces3 = cvHaarDetectObjects( transposeImage2, cascade, storage,
        1.1, 3, CV_HAAR_DO_CANNY_PRUNING,
        cvSize(40, 40) );

          face_detect3 = (faces3 ? faces3->total : 0);

    // Loop the number of faces found.
    for( i = 0; i < (faces3 ? faces3->total : 0); i++ )
    {   // Create a new rectangle for drawing the face
        CvRect* r = (CvRect*)cvGetSeqElem( faces3, i );
        // Find the dimensions of the face,and scale it if necessary
        pt1.x =320 - (r->y*scale);
        pt2.x =320 - ((r->y+r->width)*scale);
        pt1.y = r->x*scale;
        pt2.y = (r->x+r->height)*scale;
        // Draw the rectangle in the input image
        cvRectangle( img, pt2, pt1, CV_RGB(255,0,0), 3, 8, 0 );
```

```
                    }
        }
    // }
     // Show the image in the window named "Fallen Angel"
     cvShowImage( "Fallen Angel", img );

        count_2 = count_1;
        count_1 = count_now;
        count_now = 0;

        countr_2 = countr_1;
        countl_1 = countl_now;
        countl_now = 0;

        countr_2 = countr_1;
        countr_1 = countr_now;
        countr_now = 0;

        cout<<face_detect1<<"   "<<face_detect2<<"   "<<face_detect3<<endl;

        if (face_detect1 !=0) {
                    count_now = 1;

                    if (count_now == 1 && count_1 == 1 /*&& count_2 == 1*/){
                           cvSaveImage("face_found.jpg",img);
                           system("BlueSmirf_v2.exe");
                           system("Email_Sender_V3.exe");
                           found=1;
                           }
               }

        else if (face_detect2 !=0) {
                    countl_now = 1;

                    if (countl_now == 1 && countl_1 == 1 /* && countl_2 == 1*/){
                           cvSaveImage("face_found.jpg",img);
                           system("BlueSmirf_v2.exe");
                           system("Email_Sender_V3.exe");
                           found=1;
                           }
               }

        else if (face_detect3 !=0) {
                    countr_now = 1;

                    if (countr_now == 1 && countr_1 == 1 /*&& countr_2 == 1*/){
                           cvSaveImage("face_found.jpg",img);
                           system("BlueSmirf_v2.exe");
                           system("Email_Sender_V3.exe");
                           found=1;
                           }
               }
     cvReleaseHaarClassifierCascade( &cascade );
     cvReleaseMemStorage( &storage );
     // Release the temp image created.
     cvReleaseImage( &temp );
}
```

```
/*****************************************************************
 * BlueSMiRF Code
 * Created: 10/31/2012 8:37 PM
 * Author: Daniel Frank
 *
 * Description: This program opens the port to the BlueSMiRF and sends a
*                    signal to the robot that it has found a face
*****************************************************************/

#using <System.dll>
#include "stdafx.h"

using namespace System;
using namespace System::IO::Ports;
using namespace System::Threading;

int main()

{
        for(int i = 0; i < 3; i++){//sends message to stop three times
        SerialPort^ serialPort = gcnew SerialPort(L"COM7",115200,Parity::None,8,StopBits::One);

   serialPort->Open();//opens bluesmirf port

   serialPort->WriteLine("1");//signal for the robot to stop its motors

   serialPort->Close();//closes bluesmirf port
        }
   return 0;
}
```