Multi-Disciplinary Senior Design Conference
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, New York 14623

Project Number: P11216

WANDERING CAMPUS AMBASSADOR (PART 6)

Nick Leathe (*Team Lead*)
Mechanical Engineering

Anna Gilgur
Mechanical Engineering

Terra McAndrew
Industrial Design

Rui Zhou
Electrical Engineering

Ken Hertzog
Computer Engineering

Joseph Stevens
Software Engineering

Philip Gibson
Software Engineering

Dave Ladner
Software Engineering

ABSTRACT

The main purpose of the P11216 project team is to fully define, develop and implement the software programming to make the Wandering Campus Ambassador move and take care of the plant autonomously. Additional objectives included working with P11216 (Wandering Campus Ambassador (part 4 of n)) to complete and integrate the electrical, computer, and mechanical aspects of the build, ensure the robot has safety and reliability issues resolved, and prepare the robot for the ImagineRIT event in May 2011. All of the customer needs that were of the highest importance were met, appropriate safety and reliability mitigation aspects were implemented, and the Wandering Ambassador navigated and took care of the plant autonomously.

NOMENCLATURE

Angstrom-Linux operating system
Beagleboard-Single board computer used for majority of data processing
HTML-HyperText Markup Language
I2C-Inter-Integrated Circuit
Java-Programming Language
JNI-Java Native Interface
GPS-Global Positioning System
GUI-Graphic User Interface
MSP430- Microcontroller made by Texas Instruments
MySQL-Database management system
OS-Operating System
PCB-Printed Circuit Board
PHP-PHP: Hypertext preprocessor
QNX-Unix-like operating system
RIT- Rochester Institute of Technology
UML-Unified Modeling Language
XAMPP-Cross Platform, Apache HTTP Server, MySQL, PHP, Perl

PROJECT BACKGROUND

The mission of the project and Wandering Campus Ambassador is to create a robot that raises the campus awareness of self-sustainable energy by taking care of and being a guardian to a plant. The overall goal of the project is to make maximum use of natural conditions, such as sunlight/shade, temperature, and water, to take care of the plant and allow the robot to be self-sustaining. There were five other groups that worked on this project previously. They were P10215, P10216, P10217, P10218, and P11215.

P10215 and P01216 were the first two teams to work on the Wandering Campus Ambassador project and worked during the fall and winter terms of the 2009 academic school year. P10215, called “Robot Locomotion and Plant Platform”, was responsible for the shell design, frame, motors, motor drive train, plant, MSP430’s for the motor, plant and navigation sensors, plant electronics and MSP430 software.[1] P10216, called “Robot Navigation Plant Platform”, was responsible for the Beagleboard, Java OS, I2C serial interfaces, GPS, accelerometer, wireless to remote server, and the initial autonomous software.[2]

The next two teams to work on the Wandering Campus Ambassador project were P10217 and P10218. They worked on the project during the winter and spring terms of the 2009 academic school year. P10217, called "Robot Integration and Field Testing", was responsible for the body shell build, wireless game pad, refining the motor control software with motor encoders, motherboard PCB, power, and interface protocol documentation updates.[3] P210218, called "Robot Applications", was responsible for UML architecture/class diagrams, HTML, PHP, JNI, XAMPP for Windows, RIT server interface, webcam, autonomous software, sonar sensor debugging, and social media integration.[4]

P11215, the fifth team to work on the Wandering Ambassador, worked during the fall and winter terms of the 2010 academic school year. P11215, called "Wandering Campus Ambassador (Part 4 of n)", was responsible for refining and redesigning the mechanical, electrical, and software functions. [5]

In addition to fully implementing the software programming and making the Wandering Ambassador fully autonomous, this group, P11216, improved the range of the sonar sensors, added a physical deterrence to the alarm system, built holders and allocated space for additional plants, resolved safety and reliability issues, integrated all aspects of the Wandering Ambassador together, and prepared the robot for the ImagineRIT event. [6] Figure 1 is the completed Wandering Ambassador with the P11216 team.

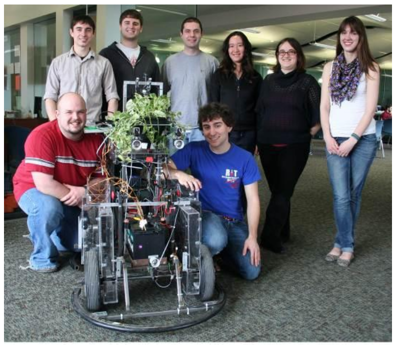
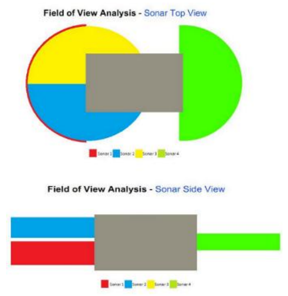


Figure 1 - Completed Wandering Ambassador

MECHANICAL DESIGN

Servo/Sonar Holders

The sonar sensors are responsible for the detection of objects and safe robot navigation, so that the Wandering Campus Ambassador does not run into objects. The placement was optimized from previous teams to provide as large a field of view as possible. There are four sonar sensors that were all be mounted on servo motors and rotate 180°. There is one located on the back of the robot, and three located on the front. Two of the ones in the front are located towards the top of the robot chassis, on either side of the plant holder, and one is located towards the bottom of the chassis. The placement of the sonars and servos on the top, front of the Wandering Ambassador lead to the impression that it has a face, which was a customer need for previous teams. Figure 2 is a field of view analysis for the placement of the sonars and servos. Figure 3 is an image part that was made to attach the servos to the sonars. Figure 4 is of the sonars and servos on the front of the Wandering Ambassador. Finally, Figure 5 is of the sonar and servo that is located on the back of the Wandering Campus Ambassador.


Field of View Analysis - Sonar Top View
Field of View Analysis - Sonar Side View
Figure 2 - Field of View Analysis for Placement of Sonars and Servos

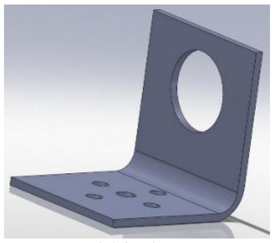


Figure 3 – Part Model for the Servo-Sonar Bracket

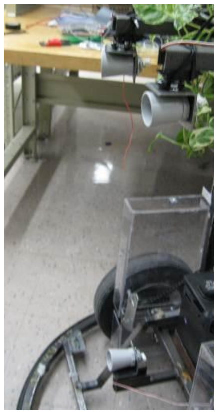


Figure 4 – Sonars/Servos Located at the Front of the Wandering Ambassador

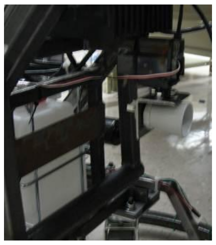


Figure 5 – Sonar/Servo Located at the Back of the Wandering Ambassador

Physical Deterrence System

Integrated by the previous teams was an alarm system. This alarm system activates when the plant is removed from its location and serves to act as an anti-vandalism measure. When P11216 received the project, the alarm system was auditory in nature. The idea was proposed to the customer to include a physical deterrence as well, of which the customer approved. The thought behind the physical deterrence was to use what was already present on the robot, the water. An additional reservoir and pump were installed on the Wandering Ambassador for the physical deterrence system. They were attached to two windshield washer nozzles that are located on the top of the chassis. The nozzles glow red when turned on, which adds to the robots "angry" personality that gets activated when the plant is stolen. Figure 6 is of the reservoirs and pumps on the Wandering Ambassador. Figure 7 is of the nozzles on the Wandering Ambassador, Figure 8 is of the nozzles glowing red.

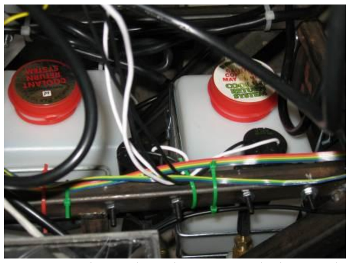


Figure 6 - Reservoirs and Tanks on Wandering Ambassador

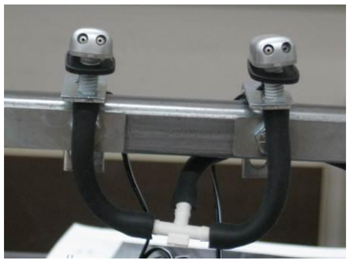


Figure 7 – Nozzles on the Wandering Ambassador

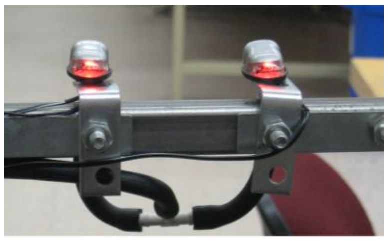


Figure 8 - Red Glowing Nozzles on the Wandering Ambassador

Additional Plant Holder

The customer requested that additional plants be added to the robot in order to promote the technology and environment integration. While currently the Wandering Ambassador is not able to take care of these plants and the main plant located on the front, future iterations of the project can make the Wandering Ambassador water and monitor the additional plants as well. It was determined that the best location for the additional plants was on top of the gears and driveshaft since there was a large, open space there. The plant holders were made and velcro'd on top of the gear cover so that if needed, it will be possible to easily remove the additional plants. The plants grew on a structure that was constructed out of reeds and attached to various points along the side of the robot. Figure 9 is an image of the additional plant holders. Figure 10 is the additional plant holders with plants on the Wandering Ambassador.

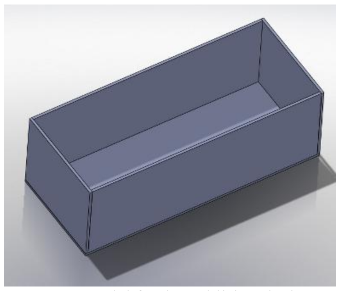


Figure 9 – Part Model for the Additional Plant Holder

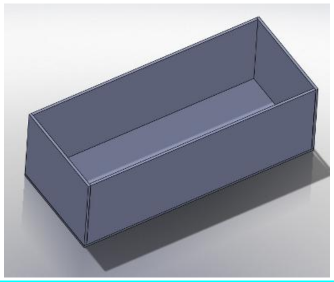


Figure 10 - Additional Plants on the Wandering Ambassador Change this picture!!

ELECTRICAL AND COMPUTER DESIGN

The electronics are all stored inside a box located towards the back of the Wandering Ambassador. Figure 11 is a 3D CAD model of the location of the electronics box. Figure 12 is an image of the wiring inside the box.

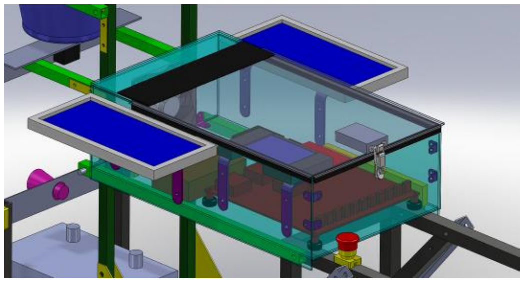


Figure 11 - 3D CAD Model of Electronics Box

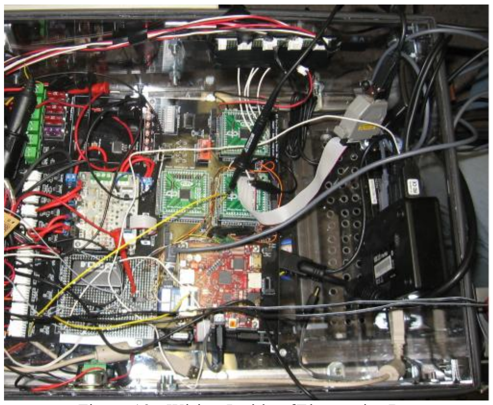
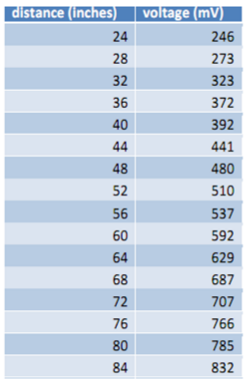


Figure 12 - Wiring Inside of Electronics Box

Sonars/Servos

The sonar range was tested indoors in the electrical engineering laboratory. The voltage that the sonars could read was tested up to a range of 7 feet. A linear relationship was found between the distance and the voltage read by the sonar. A voltage reading indicated that the sonar identified an object in its range of vision, which is two feet diameter. The sonars were attached onto the robot and wired to the beagleboard. The servos were soldered to the protoboard, which was then attached to the beagleboard. Table 1 is the collected distance and voltage measurements. Figure 13 is the beagleboard.



distance (inches)	voltage (mV)
24	246
28	273
32	323
36	372
40	392
44	441
48	480
52	510
56	537
60	592
64	629
68	687
72	707
76	766
80	785
84	832

Table 1 - Collected Distance and Voltage Measurements

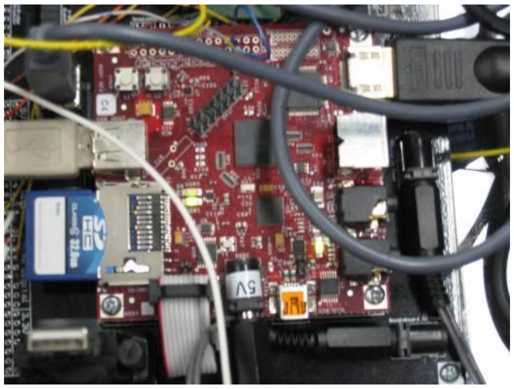
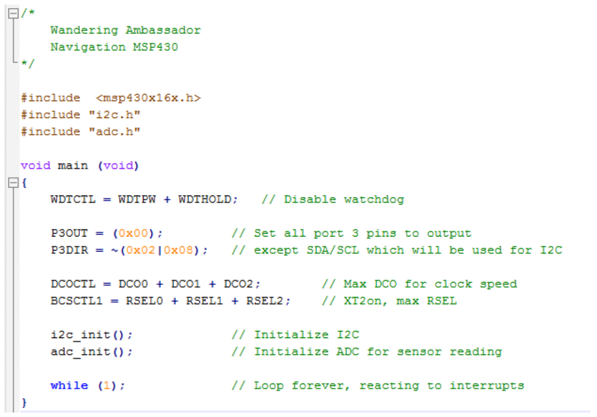


Figure 13 - Beagleboard

MSP430s

The code that existed for the MSP430s from prior teams proved to be unmanageable. It was poorly constructed, had inconsistent coding standards, and was not properly documented. This quarter, the code was rewritten so that it worked properly and would be more maintainable in the future. Figure 14 is sample code for the MSP430s. Figure 15 is an image of the MSP430s.



```
/*
    Wandering Ambassador
    Navigation MSP430
*/

#include  <msp430x16x.h>
#include "i2c.h"
#include "adc.h"

void main (void)
{
    WDTCTL = WDTPW + WDTHOLD;   // Disable watchdog

    P3OUT = (0x00);          // Set all port 3 pins to output
    P3DIR = ~(0x02|0x08);    // except SDA/SCL which will be used for I2C

    DCOCTL = DCO0 + DCO1 + DCO2;        // Max DCO for clock speed
    BCSCTL1 = RSEL0 + RSEL1 + RSEL2;    // XT2on, max RSEL

    i2c_init();              // Initialize I2C
    adc_init();              // Initialize ADC for sensor reading

    while (1);               // Loop forever, reacting to interrupts
}
```

Figure 14 - Sample Code for the MSP430s

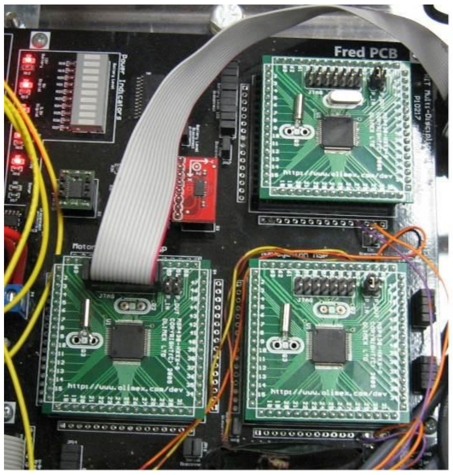
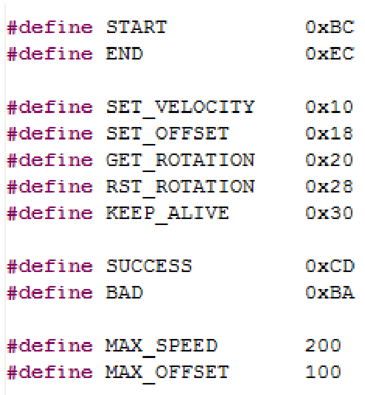


Figure 15 - MSP430s

Motor and Plant Controllers

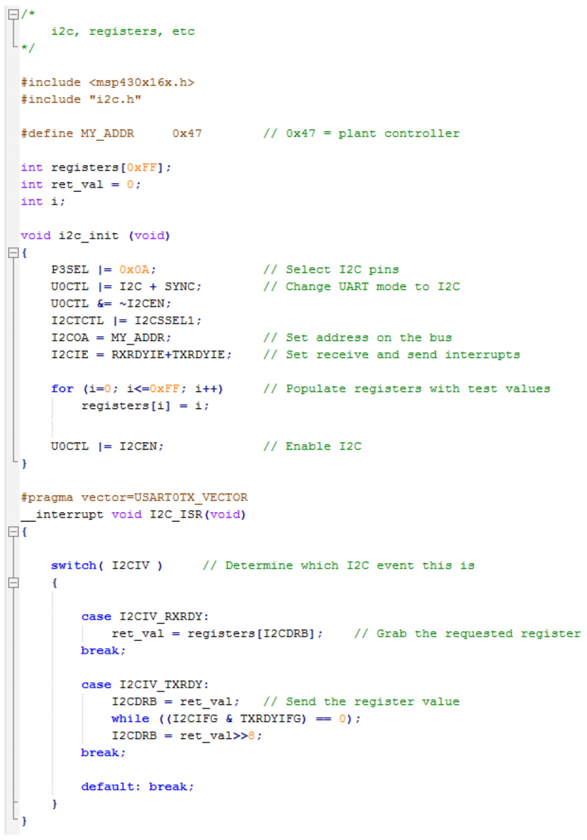
The old motor controller code had a complicated protocol that was not fully implemented. It also had bugs in the PWM code that caused an inconsistent duty cycle which performed less than optimally. The new controller code has proper PWM functionality, allowing for more consistent operation along with faster overall speed. The new protocol is simpler while maintaining all required functionality, allowing for easer implementation by the SE team. Figure 16 is an example of the current protocol being implemented.



```
#define START          0xBC
#define END            0xEC

#define SET_VELOCITY   0x10
#define SET_OFFSET     0x18
#define GET_ROTATION   0x20
#define RST_ROTATION   0x28
#define KEEP_ALIVE     0x30

#define SUCCESS        0xCD
#define BAD            0xBA

#define MAX_SPEED      200
#define MAX_OFFSET     100
```

Figure 16 - Example of Current Protocol

The motor and plant controllers operate off the same base code. The functionality is compartmentalized to a great extent allowing for easy maintainability. The I2C portion handles the I2C communication between the MSP430 and the Beagleboard, as well as the general register management. This is identical between the two controllers. The ADC portion has the same code base between the two controllers, which some changes for the different sensor requirements. For instance, the sonars cannot operate at the same time due to cross-talk, so the navigation ADC code cycles the power through them when reading. Figure 17 is a sample of the base code for the motor and plant controllers.



```
/*
    i2c, registers, etc
*/

#include <msp430x16x.h>
#include "i2c.h"

#define MY_ADDR     0x47        // 0x47 = plant controller

int registers[0xFF];
int ret_val = 0;
int i;

void i2c_init (void)
{
    P3SEL |= 0x0A;              // Select I2C pins
    U0CTL |= I2C + SYNC;        // Change UART mode to I2C
    U0CTL &= ~I2CEN;
    I2CTCTL |= I2CSSEL1;
    I2COA = MY_ADDR;            // Set address on the bus
    I2CIE = RXRDYIE+TXRDYIE;    // Set receive and send interrupts

    for (i=0; i<=0xFF; i++)     // Populate registers with test values
        registers[i] = i;

    U0CTL |= I2CEN;             // Enable I2C
}

#pragma vector=USART0TX_VECTOR
__interrupt void I2C_ISR(void)
{

    switch( I2CIV )     // Determine which I2C event this is
    {

        case I2CIV_RXRDY:
            ret_val = registers[I2CDRB];    // Grab the requested register
        break;

        case I2CIV_TXRDY:
            I2CDRB = ret_val;   // Send the register value
            while ((I2CIFG & TXRDYIFG) == 0);
            I2CDRB = ret_val>>8;
        break;

        default: break;
    }
}
```

Figure 17 - Sample of Base Code for Motor and Plant Controllers

Alarm System Circuit

The alarm circuit wiring was updated from previous teams to include the LEDs and the pump that was connected to the physical deterrence. The schematic of the alarm circuit is located in Figure 18.

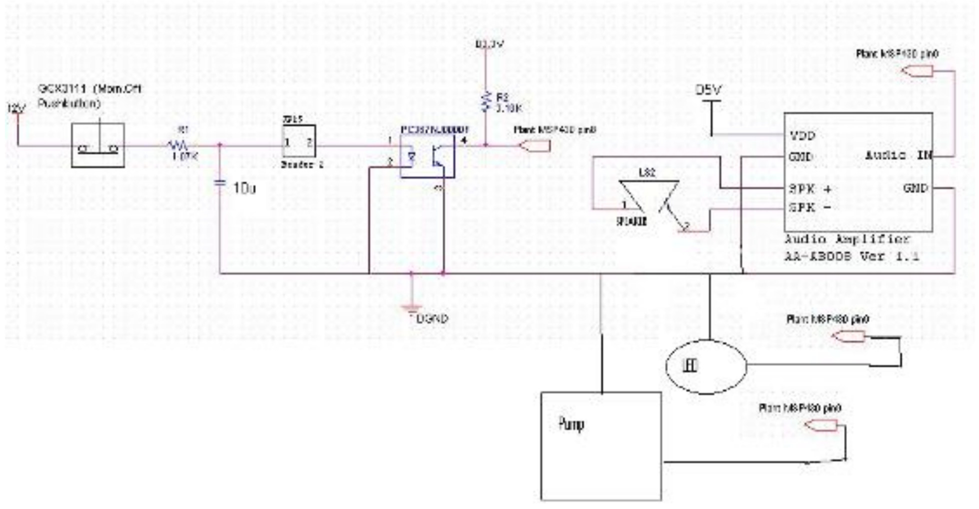


Figure 18 - Alarm System Circuit

SOFTWARE DESIGN

The software aspect of the Wandering Ambassador project was the most involved of the project. There were two main types of classes in the coding: logic classes and sensors input/output. These are outlined in further detail below. Figure 19 is the UML diagram for the software. Figures 20 and 21 are examples of sample code.

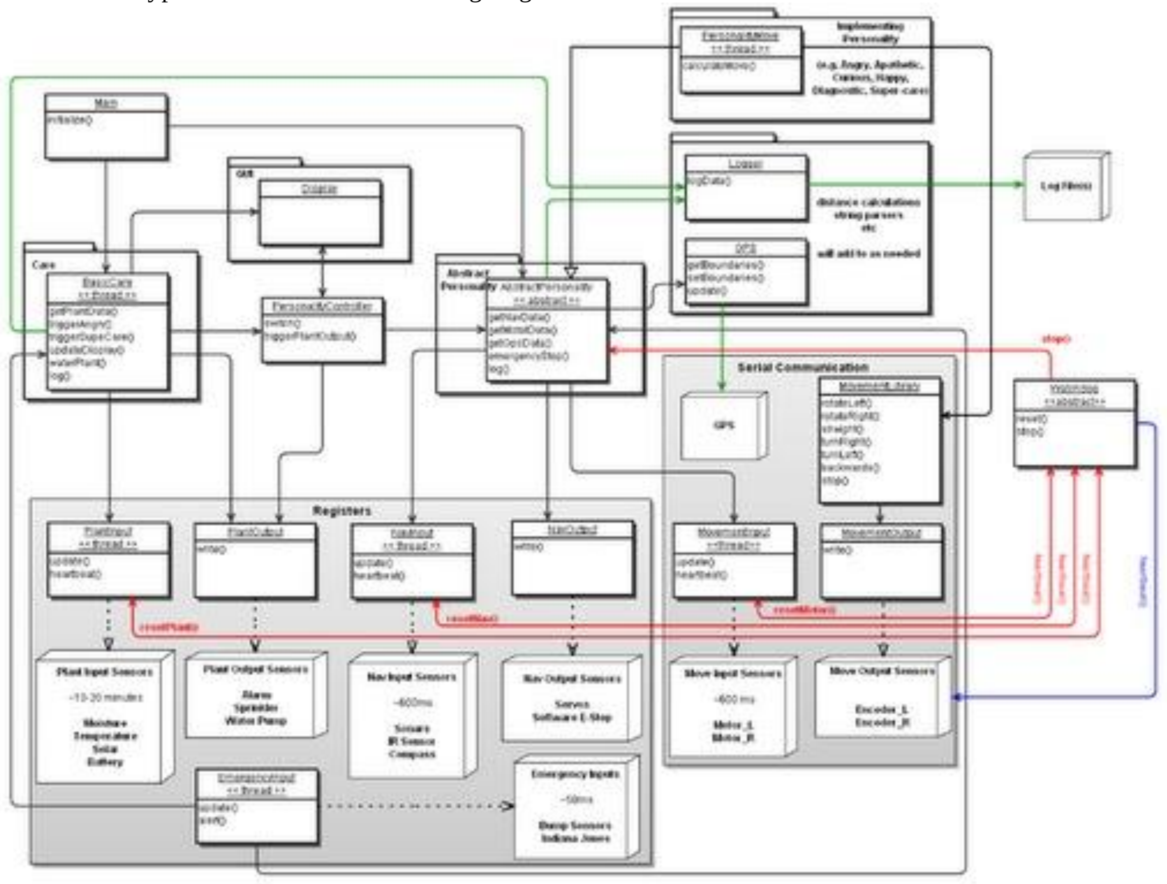


Figure 19 - UML Diagram

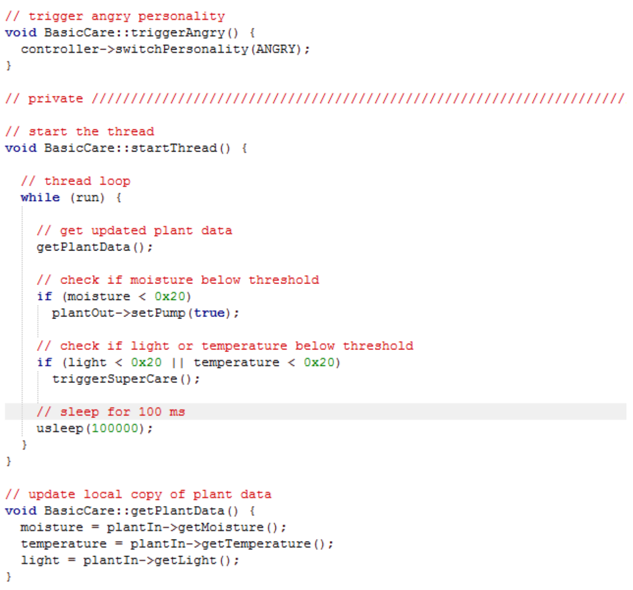

```
// trigger angry personality
void BasicCare::triggerAngry() {
  controller->switchPersonality(ANGRY);
}

// private ////////////////////////////////////////////////////////////////////

// start the thread
void BasicCare::startThread() {

  // thread loop
  while (run) {

    // get updated plant data
    getPlantData();

    // check if moisture below threshold
    if (moisture < 0x20)
      plantOut->setPump(true);

    // check if light or temperature below threshold
    if (light < 0x20 || temperature < 0x20)
      triggerSuperCare();

    // sleep for 100 ms
    usleep(100000);
  }
}

// update local copy of plant data
void BasicCare::getPlantData() {
  moisture = plantIn->getMoisture();
  temperature = plantIn->getTemperature();
  light = plantIn->getLight();
}
```

Figure 20 - Sample Code 1

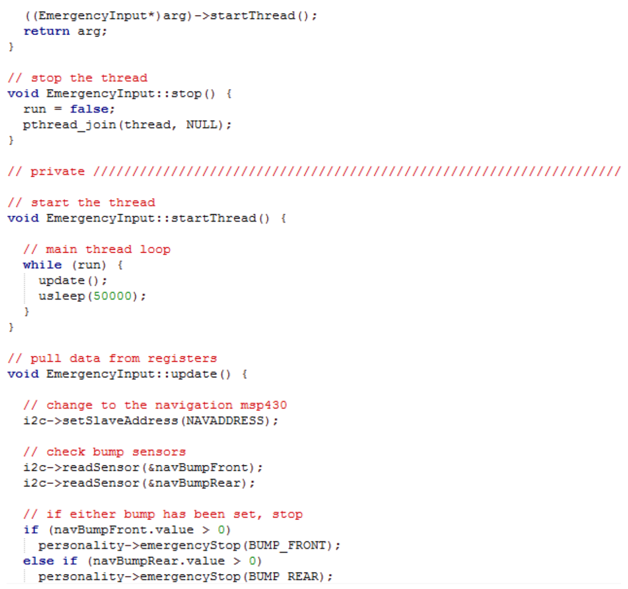

```
  ((EmergencyInput*)arg)->startThread();
  return arg;
}

// stop the thread
void EmergencyInput::stop() {
  run = false;
  pthread_join(thread, NULL);
}

// private ////////////////////////////////////////////////////////////////////

// start the thread
void EmergencyInput::startThread() {

  // main thread loop
  while (run) {
    update();
    usleep(50000);
  }
}

// pull data from registers
void EmergencyInput::update() {

  // change to the navigation msp430
  i2c->setSlaveAddress(NAVADDRESS);

  // check bump sensors
  i2c->readSensor(&navBumpFront);
  i2c->readSensor(&navBumpRear);

  // if either bump has been set, stop
  if (navBumpFront.value > 0)
    personality->emergencyStop(BUMP_FRONT);
  else if (navBumpRear.value > 0)
    personality->emergencyStop(BUMP_REAR);
```

Figure 21 - Sample Code 2

Logic Classes

There are nine logic classes. Main is an initializing class that starts the BasicCare thread and the DiagnosticMove personality thread.

PersonalityController will start and stop personality threads during runtime.

Display will display the GUI. It will receive the data that it displays from BasicCare and change color schemes with PersonalityController switches the active personality.

The utility classes represent any class that will be used to aid in calculations for the PersonlityMove. The Gps class within the utility classes receives input from the GPS, parses and translates it, and then holds for when the movement logic needs it. The Logger class within the utility classes writes log statements to files and/or output streams.

MovementLibrary contains a set of movement that the robot is able to take.

BasicCare is a thread that will check the data held in PlantInput. Based on that data it will water the plant, switch the personality, or do nothing.

AbstractPersonality is an abstract class that is meant to be overwritten by PersonalityMove. It will contain the emergencyStop() method that is inherited by all personalities.

Personality is a thread that represents the active personality of the robot. It checks the data stored in NavInput and based on the data it determines what movement should be taken. It then calls to the MovementLibrary to take that action.

Watchdog is a thread that checks to make sure that all the software and hardware interactions are still valid.

Sensors Input/Output

There are five sensor input/output classes. They are outlined in further detail below.

PlantInput is a thread that continuously checks register values and holds onto the data those registers contain. It monitors the moisture sensor, the thermometers, the solar panels, and the water level sensor.

PlantOuput writes to the registers to indicate that an action should be taken. It can command the alarm, sprinkler, or water pump. The alarm command sets the alarm off. The sprinkler command sprays water from the sprinklers, or nozzles. The water pump command waters the plant.

NavInput is a thread that continuously checks register values and holds onto the data those registers contain. It monitors the sonars, GPS, and IR sensors.

NavOuput writes to the registers to indicate that an action should be taken. It can command the servos. The servo command rotates the sonars mounted on the servos.

EmergencyInput is a thread that continuously checks register values. If the data indicates that an action needs to be taken, it will call the corresponding method to take that action in either BasicCare or CalculateMove. It monitors the bump sensors and the "Indiana Jones" switch. The bump sensor activation calls the emergencyStop() command in CalculateMove. The "Indiana Jones" switch activation calls the triggerAngry() command in BasicCare.

RESULTS AND DISCUSSION

The majority of the mechanical, electrical, and computer engineering customer needs were met. The ones that were met included: improving the sonar range and area of vision, testing the bumpers, adding a physical deterrence for the alarm, and allowing for additional plants to be carried.

Mechanical, electrical, and computer engineering customer needs that were not met were: increasing the adjustability of the plant mount, adding a switch to disengage the alarm, building a shelter for the Wandering Ambassador for the night, incorporating energy sockets for recharging the batteries within the shelter, and collecting and storing water for later use. Increasing the adjustability of the plant mount was not added due to budget and time constraints. The other customer needs that did not get implemented were not fulfilled due to time constraints.

The main focus of the work for P11216 was to deliver autonomous motion of the robot. As such, the majority of the customer needs were based on Software Engineering. The completed customer needs were storing the sensor input in the registers, developing a virtual machine for development, connecting to the Wandering Ambassador through the internet, watering the plant when the moisture level is low, staying within a certain set of coordinates, not falling off of ledges, sensing people and not running into them, apathetic, angry and diagnostic personality modes, and being able to run each motorized wheel independently.

Software engineering customer needs that were not met or were changed were: QNX loaded onto the beagleboard, seeking out more intense light when the light currently available is not enough, seeking out other plants, monitoring the battery life charge, and the curious, friendly, super-sustain, and test personality modes. QNX was not loaded onto the beagleboard because the decision was made to use Angstrom instead. The test personality mode was combined with the diagnostic personality mode, which was completed. Similarly to the mechanical, electrical, and computer engineering customer needs, the other software customer needs that were not fulfilled, were not implemented due to time constraints.

CONCLUSIONS AND RECOMMENDATIONS

The customer needs that were of the highest importance were all completed. Given more time and a larger budget, all of the customer needs, even those of low importance, could have been completed. Other setbacks include the electrical components taking longer to implement than expected and poor understanding of the hierarchy that tasks need to be completed in at the start of the project. As the project advanced, the hierarchy was better implemented and time management improved.

While there are few points to be critical of, there were several customer needs that were not met. Although these customer needs were not critical, it is the recommendation of P11216 that the customer needs that were not met be included in future iterations of the project. This includes: increasing the adjustability of the plant mount, adding a switch to disengage the alarm, building a shelter for the Wandering Ambassador for the night, incorporating energy sockets for recharging the batteries within the shelter, collecting and storing water for later use, seeking out more intense light when the light currently available is not enough, seeking out other plants, monitoring the battery life charge, and adding additional personalities.

REFERENCES

[1] Russell, Brian, Thomas Anderson, Vincent Baier, Daniel Chin, Eric Harty, Richard L. Reyes, Emile Lebrun, Manasi Manjrekar, and Matthew Padula. "P10215 / Home." *P10215: Robot Locomotion and Plant Platform.* 17 Sept. 2010. Web. 20 Apr. 2011. <http://edge.rit.edu/content/P10215/public/Home>.

[2] Alam, Maha, Corey Provencher, Alan Olson, Marcus Gwillim, and Nicolas Bouret. "P10216 / Home." *P10216: Robot Navigation Plant Platform.* 22 Apr. 2010. Web. 20 Apr. 2011. <http://edge.rit.edu/content/P10216/public/Home>.

[3] Arrigo, Pat, Wes Coleman, Aaron Zimmerman, Vernon Vantucci, Steven Guenther, and Adam Spirer. "P10217 / Home." *P10217: Robot Integration and Field Testing.* 19 May 2010. Web. 20 Apr. 2011. <http://edge.rit.edu/content/P10217/public/Home>.

[4] McAfee, Jeff, David Leeds, Katlyn Walsh, Alex Ford, and Steve Wright. "P10218 / Home." *P10218: Robot Applications.* 19 May 2010. Web. 20 Apr. 2011. <http://edge.rit.edu/content/P10218/public/Home>.

[5] Ackerman, Justin, Hersh Anand, Jonathan Notaro, Eric Tripp, Chris Hoerbelt, Gregory Mucks, and Anthony Poli. "P11215 / Home." *P11215: Wandering Campus Ambassador (part 4 of N).* 21 Feb. 2011. Web. 20 Apr. 2011. <http://edge.rit.edu/content/P11215/public/Home>.

[6] DeBartolo, Elizabeth, George Slack, Andreas Savakis, and James Vallino. "P11216 Project Readiness Package." *P11216 Project Description.* 11 Feb. 2011. Web. 20 Apr. 2011. <http://edge.rit.edu/content/P11216/public/Project%20 Description>.

ACKNOWLEDGEMENTS

Professor George Slack, Dr. James Vallino, Dr. Mario Gomes, Dr. Mark Kempski, Dr. Wayne Walter, William Spath, Professor Scott Hawker, Dr. Ken Hsu, Stan Rickel, Daniel Phillips, Steve Kosciol, Rob Kraynik, P11215, RIT Provost's Learning Innovations Grant, KGCOE EE Department, Imagine RIT Staff, Christine Fisher, RIT, Amazon.com, Robot Shop, PepBoys, Lowe's, Newark.com, RIT machine shop, MP3Car.com, Cdiweb.com, McMaster Carr, Gro-Moore Farms