

Formal Robot Proposal

**D.A.V.I.S.
Defensive Anti-Velociraptor Integrated System**

**Submitted by:
Jonathan Coad**

**EEL 5666C
Intelligent Machine Design Lab
December 9, 2009**

**Instructors:
A. Antonio Arroyo,
Eric M. Schwartz**

**Teaching Assistants:
Mike Pridgen
Thomas Vermeer**

Table of Contents

II.	Abstract.....	Page 3
II.	Executive Summary.....	Page 4
III.	Introduction.....	Page 5
IV.	Integrated System.....	Page 6
V.	Mobile Platform.....	Page 8
VI.	Actuation.....	Page 9
VII.	Sensors.....	Page 11
VIII.	Behaviors.....	Page 12
IX.	Experimental Layout and Results.....	Page 13
X.	Conclusions.....	Page 14
XI.	Documentation.....	Page 15
XII.	Appendices.....	Page 16

Abstract

The Robot called the Defensive Anti-Velociraptor Integrated System, or “D.A.V.I.S.” is a four-legged walking robot created for the purpose of hunting and eliminating predetermined targets. The balloon targets will be tracked by the robot’s on-board CMU camera using basic blob detection and color centroid determination, and a series of servo control algorithms will be used to move the robot within range of the target. Each leg has three points of articulation controlled by an independent servo and thus allowing very precise movement control. Due to the significant weight of the body structure, twelve metal-gear servos and battery pack, the robot is unable to sustain a walking motion with just the four legs, so it is additionally supported by a three-wheeled dolly mounted below the platform supported by three heavy gauge springs.

Using medium-range infrared sensors the robot performs basic obstacle avoidance. Two are mounted on the front of the robot at 45 degree angles to the direction of motion. This allows for obstacle detection in front of and also to the side of the robot. Various gaits are used to avoid obstacles depending on the sensor’s input data.

Executive Summary

The Defensive Anti-Velociraptor Integrated System, or D.A.V.I.S. for short, was designed originally to track colored objects (balloons) and destroy them- all while performing obstacles avoidance based on multiples sensor input data and various walking gaits. Ultimately due to time and monetary constraints, this was not fully achieved, although most aspects of the design were at least independently realized.

The goal behind the creation of D.A.V.I.S. was to develop a robot that resembles and reminds the observer of an animal or insect rather than just a robot. Giving the robot legs for propulsion would set it apart from the other robots and allow it to seem more 'alive'. Due to problems with total weight and weight distribution, D.A.V.I.S. is unable to support himself with just his legs alone, so he uses a rolling spring-mounted platform with ball-bearing casters as support. This ultimately allows for much greater freedom of motion and gait selection. Under no sensor constraints, the legs move such as to propel the robot forward with a certain angle of rotation-equivalent to turning a corner in a car, it turns about a variable radius as it moves forward. Supplying a value of zero to this walking function results in the robot moving straight forward.

Two medium range infrared proximity sensors are mounted at the front of the platform and face outward at 45 degree angles. These detect medium range obstacles and are used to prevent the robot from colliding with its surroundings. If the robot detects a long range obstacle on the right, it will turn left as it moves forward and vice-versa. If an object is detected close by on one side, it will stop, rotate in place to avoid it, and then continue forward. If D.A.V.I.S. finds an obstacle at medium range directly in front of it, it will choose a random (but sufficiently large) angle to rotate away and then continue walking forward. If the object is very close to it, it will reverse its gait and walk backward several steps, then rotate and walk away.

The CMU1 camera, though never fully integrated and mounted on the robot, was partially tested using a PC interface and then by feeding test coordinated to the robot. The camera would detect the center of a color blob (red balloon) and feed the x-coordinates of it to the microprocessor controlling D.A.V.I.S. Using these coordinated, the robot could rotate to center itself with the target, and then move toward it. When the combination of the CMU camera and IR sensors detected the target was in range, the robot moves into 'attack mode' in which the front legs are extended to squeeze and pop the target balloon. This feature was never fully realized due to the inability to achieve serial communication between the CMU camera and the PVR board, but will be achieved in the continued future work on the project.

Introduction

According to the online web-comic *xkcd.com*, assuming that the depictions of velociraptors are accurate in the movie *Jurassic Park*, they would be extremely dangerous to humans. In the event that an insane group of scientists is able to somehow recreate this dangerous beast lost to the world of the past, it would be helpful to have a robot to protect against them. This is the purpose of D.A.V.I.S.- the Defensive Anti-Velociraptor Integrated System. He will seek out and destroy enemy raptors* in order to protect humans.

The objective of this project is to build a small-scale prototype robot that will model a potentially much larger unit that would be able to defend against a raptor versus humans attack. The robot must be able easily walk around, have the ability to track target from reasonable range, and destroy them.

This paper will provide a breakdown of the technical specifications of the construction of the robot with detailed explanations for choice of design, parts and features. The microprocessor and its relation to and control of all other components will first be discussed. Second, the physical shape and construction of the robots main structure will be discussed. This will be followed by a description and analysis of the actuation and general movement of the robot. This includes a general discussion of various predetermined gaits and their applications during a demonstration. After this, there will be information regarding the use of various sensors as inputs. The purpose for and choices for each sensor will be explained in detail. Following, the various gaits of the robot will be discussed in greater detail along with other basic behavioral algorithms that will be followed by the microprocessor. The final results of the completed robot will be described under the 'Conclusions' section.

* All of them are enemies.

Integrated System

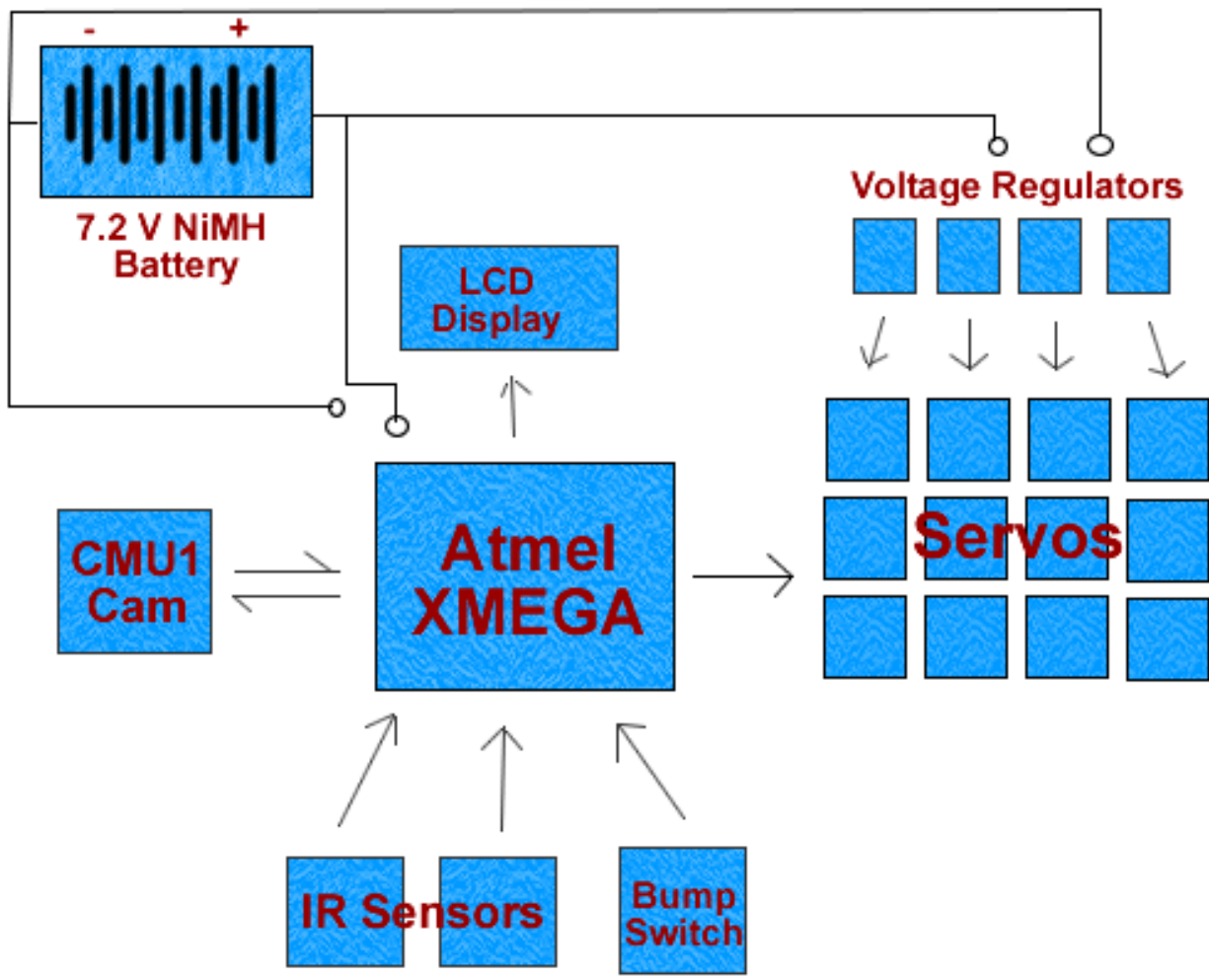
Using the development board designed and built by the TA's Mike Pridgen and Thomas Vermeer along with the provided Atmel microprocessor, the basic 'brain' of the robot will be developed and mounted on a mobile platform. The use of this chip and board will provide twelve total PWM channel, all of which are used for control of the servo motors that move the twelve leg joints. The built-in power regulators provide sufficient power to run the board, but due to the large current draw from the high-torque servos, four additional 1 A voltage regulators are attached via a separate mounted board to power the servos.

A bump sensor input will come from pull up resistors tied to a standard digital input pin. The IR sensors are tied into A/D input pins. The CMU camera is set up to communicate serially with the processor either over an RS-232 or I2C communications line, though this has not yet been implemented. Any additional sensors (more IR sensors, laser, or sonar) can easily be tied to the remaining A/D channels.

For output an LCD screen is mounted on the robot and tied to parallel 4-bit output pins from the board. In the future, arrays of various colored LED's may be tied to additional output pins to control the 'glowing light' that would come from the robot. Obviously for aesthetic purposes only, they could be used as a more clear visual indicator of the robots current state.

The microprocessor was chosen due to its large number of input/output pins as well as the large number of onboard A/D converters. Without all of this, a robot on this scale would not be possible. The speed of the processor also allows it to handle many input signals and decipher them and react accordingly all within a very short period of time. To the observer it seems instantaneous.

On the following page is a graphical representation of the connectivity of the entire system.



Mobile Platform

The main body of the robot consists of a round platform 8 inches wide. This is larger than originally anticipated, but was necessary in order to be able to fully mount all of the needed electronics. The entire body structure was made from Plexiglas brand ¼” polymer. Due to time constraints it was not cut electronically, but instead was cut by hand from a table jigsaw based on paper patterns produced from solidworks. All holes were hand drilled and fine-tuned sanding was done to help ensure symmetry. Everything was assembled with machine screws and plastic epoxy when needed.

The legs extend first up and outward, and then bend downward. This will give the robot additional girth (about 18 inches total, depending on the gait position) and height. It was originally hoped that the extension of the legs off the ground would enable it to climb, but it turns out that it does not have the necessary torque to do so. Since the platform was made of Plexiglas which is both much thicker and denser than balsa wood, the weight of the body increases significantly. This is somewhat balanced by the fact that Plexiglas is much stiffer than balsa wood, which limits the unwanted flexing the legs might otherwise have when walking. Also, higher-torque servos use metal gears rather than lightweight nylon. This further adds to the weight problem.

As it turns out, the robot is able to hold itself up quite reliably without assistance, but is unable to reliably walk without servo failure. During initial testing a 4.5 A power supply was used rather than a battery pack so as to avoid constant recharging. Often times the robot would take a step forward, stumble slightly, and the enormous sudden strain would cause some of the servos to draw upwards of 1 A each. Since the power supply could not supply this current without a voltage drop, the board would lose sufficient voltage and reset. Even if batteries were used, I was fearful that they would overheat if too much current was constantly drawn, so the decision was made to give the robot ‘training wheels.’

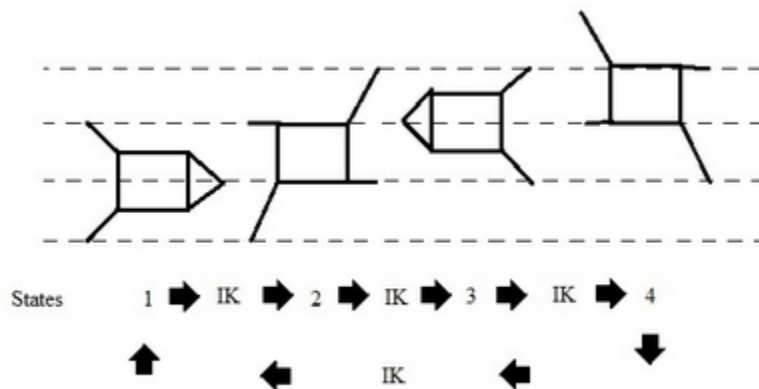
The robot now sits atop a tri-wheeled low friction rolling platform. This allows it to no longer require its legs for support, but propulsion only. Although the solution is not aesthetically pleasing as was hoped for, it does conserve a significant amount of power, as well as allows for more easily developed gaits.

Actuation

Movement of the leg joints is performed by twelve servo motors in total. Each of the four legs has a Futaba S3010 servo mounted to a special DDP110H belly pan bracket, which is mounted on the bottom of the main platform and give each leg rotation about the main axis parallel to the ground. The leg extends upwards and then downward. These two joints are controlled by two S3305 high-torque ball bearing servos. All twelve servos are controlled by the included PWM channels on the microprocessor board. Due to current draw limitations, only the S3010 servos are powered by the board's built-in PWM control ports. The other eight servos are split evenly between four separate 5V 1A power regulators built on a separate breakout board. These are necessary to prevent overheating of the printed circuit board.

The servo speeds are relatively fixed and will vary based on load, but ultimately this does not matter too much. In order to keep movement as smooth as possible, tweaking of the delays between servo function calls is necessary. Since the servos are highly accurate in terms of their positioning- and this is what truly matters when controlling a walker- making the robot go 'straight' or in a particular direction or rotation angle was not too difficult to accomplish, once the walking itself was complete, of course.

The actual walking algorithm itself was provided thanks to a fellow classmate, Seon Kim, who showed me that the walking pattern of a four legged walker could be represented by a simple state machine. By manipulating both the states themselves and the transitions between them, the robot easily be made to walk straight forward, backward, side to side, or even rotate about an axis with a variable radius. Using the concept diagram from his website:



Courtesy of Seon Kim, <http://sites.google.com/site/projectcrawlbot/design>

Changing the angles of the legs in such a manner that the points of contact (the 'feet') are always moving in a linear fashion, the robot can be made to walk straight relatively smoothly.

Reversing the pattern allows it to walk backwards and increasing the translational distance of the legs on one side relative to the other will allow it to rotate as it walks. Using a simple algorithm of my own, I also enabled D.A.V.I.S. to rotate in place without moving forward or backwards.

The greatest set back with actuation was when the parts were first ordered. Since high-end servos were chosen they were quite expensive (~\$35 a piece) and I made the mistake of assuming that (due to a misleading picture and lack of experience) the belly pan mounting brackets for \$27 a piece actually included the servos for which they were designed. They did not. Not only did this cost me another \$115, but it took time before they were finally delivered to me. Ultimately it worked out though, and the mounting brackets seem to have been a good choice.

The next problem I encountered was developing a working walking algorithm. As described above, the basic concept of using a state diagram to control all four legs simultaneously as one unit was given to me by a classmate, Seon Kim. Once this breakthrough was made and the heavy lifting completed, dropping code functions into the main behavioral loop was quite easy to complete.

Sensors

A variety of sensors are used with this robot in order to provide control information to the microprocessor. A bump sensor starts the robot after it is turned on. As it moves about it will use a CMU camera mounted on the front of the main platform in order to determine target locations. A series of movement algorithms will move it towards the target using constant camera feedback to adjust its course when needed.

A total of twelve IR sensors were purchased with the original intent that they would be strung together in a vertical fashion to somewhat 'paint a picture' of an upcoming obstacle and determine if it was tall or short. Since it is now clear that the robot will be unable to climb objects, the sensors are used in a more conventional arrangement, but in the future may also be 'stacked' close together for better averaging. Since IR sensors- like any analogue sensors- are inherently noisy, the values of several sensors are averaged over several reading cycles before deciding what the 'actual' value should be. This seems to help slightly in accuracy of the distance measuring.

The most significant problem encountered is the fact that the connector pins for the IR sensors are non-standard. This is resolved by ignoring them and simply soldering directly to the PCB.

The special sensor for this project is the CMU cam which will use blob detection and position tracking to find and track targets of a given color. Due to various unfortunate delays the CMU cam does not yet communicate with the PVR board, but development will still continue even though the project due date has passed. The following algorithm will be broken down and coded into the robot:

- scan area for blobs of a predetermined, bright color (likely orange)
- if none found then move robot according to behavioral patterns, repeat step above
- if one blob is found, determine its center coordinates relative to the camera's mid-point reference
- robot will move to position itself such that the blob is approximately centered
- repeat if not within a certain tolerance
- determine the size of the blob. Since all blobs will be the same size, the apparent size can be used to determine approximate distance from the camera
- if too far away, walk towards it
- if close enough, initialize 'raptor crushing mode'
- repeat scan
- if instead *more* than one blob is found, move towards the one already closest to the center of view, then repeat the above steps as would be done for a single blob

This may be changed depending on the capabilities of the CMU cam, although it is expected to be a fairly decent representation based on data already gathered.

Behaviors

The basic obstacle avoidance routine is as follows:

There are several cases based on averaging and constant updating of the IR sensors:

Case I: Objects are far away.

- 1.) Walk forward.

Case II: Object on the right.

- 1.) Stop.
- 2.) Rotate left twice
- 3.) Walk forward

Case III: Object on the left.

- 1.) Stop
- 2.) Rotate right twice
- 3.) Walk forward

Case IV: Object in center and/or on left and right sides

- 1.) Stop
- 2.) Backup three steps
- 3.) Rotate 90 degrees
- 4.) Re-check sensors

The robot will obey the following routine in order to carry out its objective once the CMU cam is mounted. Until then, it performs only obstacle avoidance:

- Remain in 'sleeping' mode once turned on until the operator 'bumps' the robot, waking it up.
- The robot turns around 360 degrees slowly while using the CMU camera to scan the area for target raptors.
- The first one found will be tracked. If multiple points are found on the same screen shot then the center-most blob will be followed.
- The robot will move toward the blob while maintaining it at the center of the cameras point of view
- The robot will move to the side some and re-check to see if it is still there
- Repeat several times and if the object is not cleared or another object is encountered, then it will rotate 180 degrees and continue obstacle avoidance while searching for targets
- If at any time during this the robot loses track of the target blob, it will rotate at 90 degrees away from the obstacle and re-search for targets
- If the obstacle can be climbed then it will be
- After the obstacle is cleared the robot will continue to move toward the blob
- The robot will move closer to it, rotate slightly, and then crush the target with its massive feet, popping the balloon raptor target
- The robot will then continue searching for additional targets and repeat the entire process

Experimental Layout and Results

The first experiment performed, out of order, was the robot standing up. Using the servos alone without the platform, and being powered by an external power supply, the robot was able to hold itself up, as well as up to a 5 lb added weight. Unfortunately, this was only made possible due to the angle of the legs- with the furthest part of the extremities at anything shallower than about 80-85 degrees; it was unable to stand at all. This meant that the actual act of walking would be near impossible without more support, although it did take me several weeks to come to this unfortunate conclusion.

The sensor testing should have been done long before the robot platform was even made to stand, but unfortunately I decided to wait until it was working before considering the IR sensors. I noticed they seemed very effective at medium range ~20- 65 cm, but anything beyond that proved fairly unreliable, even with multiple data averaging. One of the problems I ran into here was that I unwisely chose not to order the custom-shaped cables that the IR sensors required to hook up to the PVR board directly. Instead, I chose to solder wires directly to the PCB part of the sensor, and solder that to individual connectors before connecting it to the board. While this worked for a time, the signal wire broke loose on demo day for the left sensor and I did not realize it until afterward. This resulted in unpredictable floating pin signals and D.A.V.I.S. being unable to fully perform obstacle avoidance properly.

All CMU1 Cam testing took place by successfully using windows HyperTerminal to communicate via an RS-232 serial communications line with the camera's microprocessor. Blob tracking was initiated with the camera being fed a min and max values for red color blob tracking and outputting to HyperTerminal the coordinates of the centroid of the red blob(s) detected. Unfortunately, little development successfully made it beyond this testing stage since successful serial communication was never made with between the PVR board and the camera's microprocessor.

Conclusions

Although the D.A.V.I.S. project ended in a result less than I'd hoped for and expected, it was still quite successful in many ways. In the beginning there were many extremely extravagant plans made in the which the robot would not only walk, but climb small object, fire a mounted dart blaster, light up, and include a large array of sensors that would allow it to make complex, intelligent decisions about its movement. Many of these ideas and expectations were dropped early on, then others later on due only to time and money constraints.

In total, the robot cost approximately \$1100. This was about \$500 more than expected, but I had never built anything like this before, so I didn't realize just how much certain parts ca cost. For example, the battery alone was \$35, not to mention the \$40 charger that was required for it. I was lucky enough to have not burned out any servos in testing, since they cost almost \$30 a piece. I originally planned to purchase multiple types of sensors- various range IR, sonar and even a laser range finder. To better fit my budget I ended up just buy a dozen medium range IR sensors, although further reduced plans only used two of them.

Due to weight limits there is no longer any type of dart blaster mounted on top of the robot for a 'stun' effect of shooting the target. Due to torque and power limits the robot will not be able to climb any obstacles. To carry out its main objective it instead was planned to pop balloons between its 'feet' by squeezing them with thumbtack lined legs.

If I had more time to work I would have first gotten the CMU camera fully functional and communicating with the PVR board. This would have allowed the robot to fully perform its desired goal. Beyond that, I would add eight more IR sensors, and program them to turn on and off in sequence to gain more accurate obstacle proximity without signal interference. A 180 degree arc area would be covered by five medium-range IR sensors paired with five long-range sensors. This would allow for much more complex behavior with both obstacle avoidance and target tracking.

The class is over and I have learned many more things that ever expected. Because it has been so much fun building this robot from the ground up I will continue to improve its design and programming in the years to come. Once D.A.V.I.S. is perfected, he will be rebuilt from scratch using a different design keeping in mind new and improved objectives and the lessons already learned.

Documentation

Sources for information outside of my own personal testing and development:

Mike and Thomas' PVR board documentation and library in C language

Seon Kim's state machine for quadrupeds:
<http://sites.google.com/site/projectcrawlbot/design>

CMU1 Cam's shipped pdf manual

Appendices

The following are samples of code used to test, configure and control the robot.

[initially setting up the robot]

```
#include <avr/io.h>
#include "PVR.h"

void main(void)
{
    xmegaInit();           //setup XMega
    delayInit();          //setup delay
    functions
        ServoCInit();     //setup PORTC
    Servos
        ServoDInit();     //setup PORTD
    Servos
        ADCAInit();       //setup
    PORTA analong readings
        lcdInit();        //setup
    LCD on PORTK
        lcdString("PV Robotics"); //display "PV
    Robotics" on top line (Line 0) of LCD
        lcdGoto(1,0);     //move LCD
    cursor to the second line (Line 1) of LCD
        lcdString("Board Demo"); //display "Board
    Demo" on second line
        PORTQ_DIR |= 0x01; //set Q0 (LED)
    as output
        int i = -100;
        while(1)
        {
            while(1)
            {
                ServoC0(i); //move
    ServoC0 to current position
                ServoD0(i); //move ServoD0
    to opposite of current position

                delay_ms(6000);
                ServoC0(-i); //move
    ServoC0 to current position
                ServoD0(-i);
```

```

        delay_ms(6000);
    }
    i+=10;
    if (i >= 0) //If on
positive half of servo range...
        PORTQ_OUT = 1; // turn on
LED
    else //If not on
positive half of servo range...
        PORTQ_OUT = 0; // turn off
LED
    if (i > 100) //If past
servo range, reset into range
        i = -100;
        delay_ms(100); //delay 100ms
    }
}

```

[learning to walk & LCD display]

```
#include <avr/io.h>
```

```
#include "PVR.h"
```

```
void main(void)
```

```
{
```

```
    xmegaInit();
```

```
    //setup XMega
```

```
    delayInit();
```

```
    //setup delay functions
```

```
    ServoCInit();
```

```
    //setup PORTC Servos
```

```
    ServoDInit();
```

```
    //setup PORTD Servos
```

```
    ADCAInit();
```

```
    //setup PORTA
```

```
analog readings
```

```
    lcdInit();
```

```
    //setup LCD on
```

```
PORTK
```

```
    lcdString("IMDL 2009");
```

```
    //display "PV Robotics" on top line
```

```
(Line 0) of LCD
```

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

```
    //move LCD cursor to the
```

```
second line (Line 1) of LCD
```

```
    lcdString("D.A.V.I.S.");
```

```
    //display "Board Demo" on second
```

```
line
```

```
    delay_ms(9000);
```

```
    lcdGoto(0,0);
```

```
    lcdData(1); //clear lcd display
```

```
    PORTQ_DIR |= 0x01;
```

```
    //set Q0 (LED) as
```

```
output
```

```

while(1)
{

// Robot Stands up //
    leg11(0);
    leg21(0);
    leg31(0);
    leg41(0);
    leg12(-40);
    leg22(-40);
    leg32(-40);
    leg42(-40);
    leg13(-120);
    leg43(-120);
    leg33(-120);
    leg23(-120);
    delay_ms(3000);

    lcdString("Searching...");
    delay_ms(400);
    lcdData(1);          //clear lcd display
    delay_ms(400);
    lcdString("Searching...");
    delay_ms(400);
    lcdData(1);          //clear lcd display
    delay_ms(400);
    lcdString("Searching...");

//////////
// look around a bit //
    leg11(-75);
    leg21(-75);
    leg31(-75);
    leg41(-75);
    delay_ms(2000);
    leg11(75);
    leg21(75);
    leg31(75);
    leg41(75);
    delay_ms(2500);
    leg11(0);
    leg21(0);
    leg31(0);

```

```

leg41(0);
delay_ms(2000);
////////////////////
while(1)
{
leg21(-40);
leg41(40);
delay_ms(400);
leg11(-60);
leg12(-100);
delay_ms(500);
leg12(-40);
delay_ms(400);
leg21(0);
leg41(0);
delay_ms(9000);

/*
leg42(-100);
leg43(-85);
leg22(-10);
leg23(-80);
delay_ms(500);
leg22(-90);
leg21(-60);
delay_ms(500);
leg22(-40);
leg23(-100);
leg42(-40);
leg43(-100);
delay_ms(700);

leg32(-100);
leg33(-85);
leg12(-10);
leg13(-80);
delay_ms(500);
leg12(-90);
leg11(-60);
delay_ms(500);
leg12(-40);
leg13(-100);
leg33(-100);
leg32(-40);
delay_ms(700);

leg22(-100);

```

```
leg23(-90);  
leg42(-10);  
leg43(-80);  
delay_ms(500);  
leg42(-90);  
leg41(-60);  
delay_ms(500);  
leg42(-40);  
leg23(-100);  
leg43(-100);  
leg22(-40);  
delay_ms(700);
```

```
leg12(-100);  
leg13(-90);  
leg32(-10);  
leg33(-80);  
delay_ms(500);  
leg32(-90);  
leg31(-60);  
delay_ms(500);  
leg32(-40);  
leg13(-100);  
leg33(-100);  
leg12(-40);  
delay_ms(700);
```

```
*/
```

```
leg11(0);  
leg21(0);  
leg31(0);  
leg41(0);  
delay_ms(2000);
```

```
lcdData(1);           //clear lcd display  
lcdGoto(0,0);  
lcdString("Hi there!");  
delay_ms(500);
```

```
}
```

```
}
```

```
}
```

[Leg joint configuring]

```

#include <avr/io.h>
#include "PVR.h"

void main(void)
{
    xmegaInit();           //setup XMega
    delayInit();          //setup delay functions
    ServoCInit();         //setup PORTC Servos
    ServoDInit();         //setup PORTD Servos
    ADCAInit();           //setup PORTA

    analog readings
    lcdInit();             //setup LCD on
    PORTK
    lcdString("PV Robotics"); //display "PV Robotics" on top line
    (Line 0) of LCD
    lcdGoto(1,0);         //move LCD cursor to the
    second line (Line 1) of LCD
    lcdString("Board Demo"); //display "Board Demo" on second
    line
    PORTQ_DIR |= 0x01;    //set Q0 (LED) as
    output
    int i = -100;
    int value1;
    while(1)
    {

//        value1 = ADCA0();
//        ServoC0((-100 + (value1 / 4)));
//        lcdString([value1]);
//        delay_ms(100);

        delay_ms(3000);

        ServoC1(0);
        ServoC2(0);
        ServoD2(0);
        ServoD3(0);
        delay_ms(500);

        ServoD4(-20);    //joint 2's
        ServoC4(20);
        ServoC3(-20);
    }
}

```

```
ServoD1(20);

delay_ms(750);

ServoD5(-65);           ///joint 3's
ServoC5(65);
ServoC0(-65);
ServoD0(65);

delay_ms(6000);
while(1)
{
  ServoC1(50);
  ServoC2(50);
  ServoD2(50);
  ServoD3(50);
  delay_ms(500);

  ServoC1(-50);
  delay_ms(500);
  ServoC2(-50);
  delay_ms(500);
  ServoD2(-50);
  delay_ms(500);
  ServoD3(-50);
  delay_ms(500);
}
}
```

```

#include <avr/io.h>
#include "PVR.h"
#include <avr/portpins.h>

#include "uart.h"

void main(void)
{
    xmegaInit();           //setup XMega
    delayInit();          //setup delay
    functions
    ServoCInit();         //setup PORTC
    Servos
    ServoDInit();         //setup PORTD
    Servos
    ADCAInit();           //setup
    PORTA analong readings
    lcdInit();            //setup
    LCD on PORTK
    lcdString("IMDL 2009"); //display "PV
    Robotics" on top line (Line 0) of LCD
    lcdGoto(1,0);         //move LCD
    cursor to the second line (Line 1) of LCD
    lcdString("D.A.V.I.S."); //display second
    line
    uart_init();

    delay_ms(5000);
    lcdGoto(0,0);
    lcdData(1);          //clear lcd display

    PORTB_DIR = 0x00;    //set B7 as
    input

    int normal;

    lcdData(1);          //clear lcd display
    delay_ms(300);
    lcdString("Sleeping...");

    while((PORTB_IN & 0b0000001) == 0b00000001)

    {

        delay_ms(1);

```

```

    }

    lcdData(1);          //clear lcd display
    delay_ms(300);
    lcdString("Waking Up...");
    delay_ms(1000);

normalize();
normal = 1;

    delay_ms(2000);
    lcdData(1);
    lcdString("Searching...");
    delay_ms(200);
    lcdData(1);          //clear lcd display
    delay_ms(200);
    lcdString("Searching...");
    delay_ms(200);
    lcdData(1);          //clear lcd display
    delay_ms(200);
    lcdString("Searching...");
    delay_ms(200);

    int distR1;
    int distL1;
    int distR2;
    int distL2;
    int distR3;
    int distL3;
    int distR4;
    int distL4;
    int distR5;
    int distL5;

    int distR;
    int distL;

while(1){

    ////////////////////////////////////////////////////
    // look around a bit ///
    distR1 = ADCA0();
    distL1 = ADCA2();
    delay_ms(20);
    distR2 = ADCA0();
    distL2 = ADCA2();

```

```

delay_ms(20);
distR3 = ADCA0();
distL3 = ADCA2();
delay_ms(20);
distR4 = ADCA0();
distL4 = ADCA2();
delay_ms(20);
distR5 = ADCA0();
distL5 = ADCA2();
delay_ms(20);;
distR = distR1;
distL = distL1;

//distR = ((distR1 + distR2+distR3+distR4+distR5)/5);
//distL = ((distL1 + distL2+distL3+distL4+distL5)/5);

//          uart_sendstring("\r");          //put camera into
raw transfer mode

//          unsigned char test = uart_getchar();

//          lcdString(test);
//          delay_ms(3000);
          lcdData(1);
          lcdInt(distL);

if(distL > 400 && distR > 650)
{
  if(normal == 1)
  {
    normal = 0;
    up();
  }

  walkf(0);
  walkf(0);
}
else if(distL <= 400 && distR > 650)
{

  if(normal == 0)
  {
    walkb();
    walkb();
    up();
    normalize();
    normal = 1;
  }
}

```

```

    }

    rotate(-100);
    }
else if(distL > 400 && distR <=650)
{
    if(normal == 0)
    {
        walkb();
        walkb();
        up();
        normalize();
        normal = 1;
    }

    rotate(100);
    }
else if(distL <= 400 && distR <= 650)
{
    normal = 0;
    up();
    walkb();
    walkb();
    walkb();
    up();
    normalize();
    rotate(100);
    rotate(100);
    }

} //end of while loop

} //end of main();

void state1(int value)
{
    leg21(25+(value/2));
    leg22(0);
    leg23(-50);

    leg11(-120);
    leg12(-30);
    leg13(-80);

```

```
        leg31(-25-(value/2));
        leg32(-5);
        leg33(-55);

        leg41(120);
        delay_ms(250);
        leg42(-30);
        leg43(-80);
    }
void state2(int value)
{
    leg11(-25+(value/2));
    leg12(0);
    leg13(-50);

    leg21(120);
    leg22(-30);
    leg23(-80);

    leg41(25-(value/2));
    leg42(0);
    leg43(-50);

    leg31(-120);
    delay_ms(250);
    leg32(-30);
    leg33(-80);
}
void state3(int value)
{
    leg31(15-value);
    leg32(20);
    leg33(-25);

    leg41(50-(value/3));
    leg42(-20);
    leg43(-70);

    leg21(50+(value/3));
    leg22(-20);
    leg23(-70);

    leg11(15+value);
    delay_ms(250);
    leg12(20);
    leg13(-30);
```

```
}  
void state4(int value)  
{  
    leg41(-15-value);  
    leg42(20);  
    leg43(-30);  
  
    leg31(-50-(value/3));  
    leg32(-15);  
    leg33(-70);  
  
    leg11(-50+(value/3));  
    leg12(-20);  
    leg13(-70);  
  
    leg21(-15+value);  
    delay_ms(250);  
    leg22(20);  
    leg23(-30);  
}
```

```
void walkf(int deg)  
{  
    state3(deg);  
  
    delay_ms(250);  
        leg32(-100);  
        delay_ms(200);  
  
    state2(deg);  
  
    delay_ms(250);  
        leg22(-100);  
        delay_ms(200);  
  
    state4(deg);  
  
    delay_ms(250);  
        leg42(-100);  
        delay_ms(200);  
  
    state1(deg);  
  
    delay_ms(250);
```

```
        leg12(-100);
        delay_ms(200);
    }

void walkb(void)
{
    state1(0);

    delay_ms(250);
        leg12(-100);
        delay_ms(200);

    state4(0);

    delay_ms(250);
        leg42(-100);
        delay_ms(200);

    state2(0);

    delay_ms(250);
        leg22(-100);
        delay_ms(200);

    state3(0);

    delay_ms(250);
        leg32(-100);
        delay_ms(200);

}

void normalize(void)
{
    leg11(0);
    leg12(0);
    leg13(-50);
    delay_ms(150);
    leg31(0);
    leg32(0);
    leg33(-50);
    delay_ms(150);
    leg21(0);
    leg22(0);
    leg23(-50);
    delay_ms(150);
}
```

```
        leg41(0);
        leg42(0);
        leg43(-50);
        delay_ms(150);
    }

void up(void)
{
    leg12(-100);
    delay_ms(100);
    leg32(-100);
    delay_ms(100);
    leg22(-100);
    delay_ms(100);
    leg42(-100);
    delay_ms(300);
}

void rotate(int value1)
{
    leg11(0-value1);
    leg21(0-value1);
    leg31(0-value1);
    leg41(0-value1);
    delay_ms(400);

    leg12(-40);
    leg11(0);
    delay_ms(250);
    leg42(-55);
    leg41(0);
    delay_ms(250);
    leg32(-40);
    leg31(0);
    delay_ms(250);
    leg22(-40);
    leg21(0);
    delay_ms(250);

    normalize();
}
}
```