

Vikis: The Can Collecting Robot

A Senior Project  
Presented to  
The Faculty of the Enter Computer Engineering Department  
California Polytechnic State University, San Luis Obispo

In Partial Fulfillment  
of the Requirements for the Degree  
Bachelor of Science in Computer Engineering

by

Eric Edwards  
Leah Humiston  
Jason Foulk

May, 2012

© 2012 Eric Edwards, Leah Humiston, Jason Foulk

***Abstract-RoboRodentia is an annual school-wide robotics competition that has been held during Cal Poly's Open House for the past thirteen years. The competition rules change from year to year as the objective of the competition changes, but an important aspect of the competition is that the robots run completely autonomously. This paper describes the development, design, and implementation of the RoboRodentia robot from Team Vikis. The technical specifications, robot components, individual responsibilities, and results are described in depth within this document.***

## Introduction

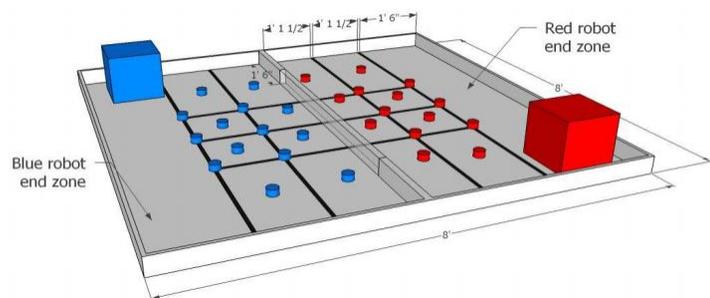
Cal Poly fosters an environment of education and hands-on learning. CPE462 is the second course in the two quarter long senior project series for computer engineers. RoboRodentia is viable senior project for engineering students and is advised by faculty member Dr. John Seng. RoboRodentia as a senior project offers students an interdisciplinary project design opportunity that involves designing a robot that meets the competition specifications, building the robot, and entering the robot into the RoboRodentia competition.

### *a. Competition Overview*

The Cal Poly RoboRodentia competition's objective is different each year but maintains several general competition guidelines that are consistent throughout the years. The 2012 objective was for robots to collect as many 3oz Fancy Feast cat food cans into their goal zone as possible during within the time limit. At the start of the competition each robot starts on their side of a field within their goal zone at any orientation the competitors chose. At the start

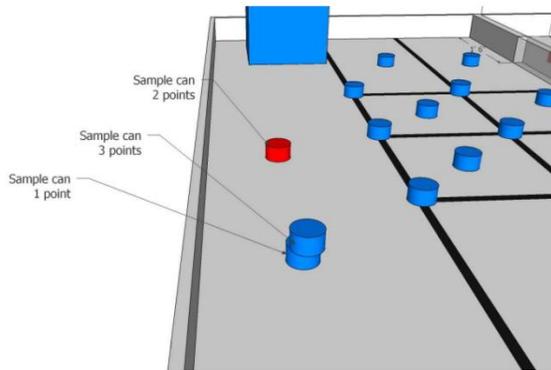
time there is a center divider dividing the two sides of the playing field. The course layout is demonstrated in Figure 1, with an 8'x1'6" goal zone on each side and a playing field of 3'x8'. Each robot has the first 60 seconds to collect and deposit as many of the 16 cans from their side as possible; after 60 seconds the center divider is lifted and the robots have to opportunity to steal cans from the opponents playing field. The cans are color coded to signify which team's playing field they originated and different cans are scored differently depending where they originated and if they are stacked when they are deposited.

**Figure 1: Course Layout**



### *b. Scoring*

A can is scored as long as it is at least partially in the goal zone and the scoring is as follows: 1 point for each can collected from the home field and 2 points for any opponent's cans in the goal zone. A can in a stack will be worth 2 points more per can in the stack. Figure 2 demonstrates a typical scoring scenario.



If both teams tie in can score, tie breakers include: robot with the tallest stack, else robot to score the most opponent cans, else robot to move more cans from the black tape intersections, else whichever robot is not in their own goal zone at the end of the match.

#### *c. Robot Specification*

The robots in the competition had to be fully autonomous as well as self-contained. The robots had to be 12"x12" or smaller at the beginning of the match and can autonomously expand up to 14"x14" during the match. The robots cannot disassemble, damage the course/contest cans, or utilize RF communication.

#### *d. Competition Regulations*

The entire match lasts 3 min, with the center divider being lifted after the first minute. Competitions can end early if both teams agree and if a member touches a robot then the run ends for that robot and the team keeps all points collected up until then. If robots become entangled for 10 seconds, a referee stops the match clock and each team must restart their robot anywhere in their goal zone. The tournament was held in a double elimination format, once a robot loses one competition it is moved into the loser's bracket and once a robot

loses two competitions it is taken out of the tournament.

#### *e. Penalties*

A robot that attempts to damage an opponent's robot will be disqualified, a robot that does not move for 60 seconds will be taken out of a match, and a robot that does not leave their goal zone within the first 20 seconds of the match will forfeit.

### **Design**

#### *a. System Design*

Our system can be broken up into three primary subsystems as seen in Fig 3: The PolyBot board Microcontroller, the motors controlling movement, and the line detection sensors mounted on the front of the robot. As you can see, the PolyBot board is the primary controller for all functions and movement the robot performs. The PolyBot board receives data from the line following sensors and outputs a voltage to the wheels. This voltage is supplied by a dedicated power source; the board only regulates this voltage, and has its own power source. The voltages (aka the movement commands) sent to the wheels are based on the data taken in from the sensors and timing; the timing is handled by the board's clock.

#### *b. Hardware design*

See Appendix A.

#### *c. Software algorithms*

The software for Vikis is broken down into three main categories. The first category is movement, and is comprised of a number of functions designed to allow Vikis to have

great freedom of movement in his environment. Some of these functions are (list a few), and a complete list of these movement functions and their descriptions can be found in Appendix C. Next are the line sensing and orientation functions, which allow Vikis to detect lines on the play field and orientate himself based on those lines. These functions and their descriptions are also found in Appendix C. The last category is the Main execution thread. This thread makes use of the other two categories functions to navigate through the play area and collect both friendly and enemy cans. The main thread contains two separate paths of execution; one contains code that does a basic can collection on the friendly side of the play area. The second path contains attack code, which sends the robot to the enemy side of the play area and steals cans from both their play area and goal zone. Upon finishing the first path Vikis begins execution of the attack code. Again the description of the main thread can be found in Appendix C.

#### *d. Mechanical design*

The Mechanical Design for Vikis is quite simple. We chose to take a simpler design to make our robot more reliable. We start with a basic wood construction, a platform upon which to place hardware and collect cat food cans. This base consists of a 5x4 rectangle that comprises the primary hardware platform and a backstop upon which cans are pushed. Attached to this platform are two arms of approximately 5.5 inches. These arms are attached using 90 degree brackets and simple screws. To optimize our collection of the cans the brackets are bent out to approximately 107.5 degrees so the arms are at a slight outward angle. The brackets were adjusted

after mounting of the arms to make sure that the arms did not violate the size limitations of the competition; therefore so the exact angle is subject to change based on the length of each arm, which can be replaced if needed. Next we used mounting washers and screws to attach the Polybot board to the base. Once the board was secured to the base we attached the motors to the underside of the base using mounting brackets designed to integrate with the motors and allow them to be screw secured into the wood. The next item to be attached was the line detecting sensors. These were attached simply with glue and tape. During their attachment they were calibrated to the correct distance from the ground to properly detect the lines of the play field. Lastly the two front ball-caster wheels, mounted on the ends of the arms. These were attached using screws and superglue. See Images 1-3 in Appendix A

#### *e. Components*

As discussed in the Mechanical design section Vikis is quite simple in its construction, and therefore does not consist of many components. His primary structure is comprised of 4 pieces of simply wood. The base is a single 5x4x1 inch rectangle and the arms and backstop for pushing cat food cans are three 1x2x.25 inch planks. Next are the electrical components, the primary controller is a PolyBot board v1.1 consisting of an Atmel ATmega644 and various inputs and outputs. The full specifications for the PolyBot board can be found in the links in the references section. There are also two 25Dx52L mm HP 47:1 Metal Gearmotors with mounting brackets and wheels. These motors operate at 6 volts with a 2 amp stall current. Our

line detection sensors consist of four digital QRE1113 Line Sensors. Each sensor has a power, and ground line to the board, as well as a data line that sends a 1 or a 0 depending on whether or not it detects a line. There are also various wires and connectors to link all the electrical devices. Links to the motors and sensors can be found in the references section. The last item are the two ball-caster wheels attached to front arms of Vikis.

#### *f. Design decisions & Evaluation*

The first real decision we made about the design of our robot was whether we wanted to stack the cans or just push them into the end zone. We had several ideas of how to implement both solutions, but decided to go with the “pushing” method. The reason we chose to go with such a basic mechanical design was just that, it was simple and straightforward. We believed that going with an overcomplicated design and dealing with stacking could have game time complications that caused the robot to fail. Based on the decision to push instead of stack, our next decision was how to do it (both how physically, and how strategically). We used “arms” on the robot going out diagonally from the edges of its base to provide maximum room to collect cans until the robot could deposit them. After the physical method was decided on, we had to choose the method of which to find cans. Since we knew where some of the cans were already going to be, we decided to not waste time implementing a solution that searched for cans. Instead, we hard coded a path for our robot to take in order to maximize the amount of cans collected and deposited, using line detecting sensors to reorient if the robot were ever to get misaligned. At this point, we had to decide

whether to try and protect the cans we collected or to go on the offensive and attempt to steal the opponent’s cans. We completed the collection on our side in less than one minute, and decided it would be better to use the extra time we had to attempt to score more points by attacking our opponents. We once again used a hard coded path to move to the opponent’s side and then back to our own with the cans.

Based on the performance of our robot, the strategy we chose worked out exactly as we expected. Many of the opponent’s robots had last minute errors that could not be repaired, causing them to lose one or both of their matches. The first minute of our strategy was very predictable, allowing us to collect the cans as expected. The second two minute section was slightly less predictable because the other robot occasionally got in the way of ours, interrupting the hard coded path. In most cases this did not occur. Below is a decision matrix explaining how we decided on our design. The rating goes from 1-5, with 1 being well average, 3 being average, and 5 being well above average. These ratings are based on how we felt before the competition, and are consistent with how other teams robots who implemented these solutions worked.

	Speed	Consistency & Reliability	Points Possible	Total Score	Rank
Pushing: hard coded path & guard cans	5	5	1	<b>11</b>	<b>2</b>
Pushing: hard coded path & offensive	5	4	3	<b>12</b>	<b>1</b>
Pushing: can finding & guard cans	3	3	1	<b>7</b>	<b>7</b>
Pushing: can finding & offensive	3	2	3	<b>8</b>	<b>5</b>
Stacking: hard coded path & guard cans	3	2	4	<b>9</b>	<b>4</b>
Stacking: hard coded path & offensive	3	2	5	<b>10</b>	<b>3</b>
Stacking: can finding & guard cans	2	1	4	<b>7</b>	<b>7</b>
Stacking: can finding & offensive	2	1	5	<b>8</b>	<b>5</b>

Figure 2: Decision matrix displaying how we decided on our design (rated 1-5).

### **Conclusion**

Our system operated as we expected and was very successful. As the decision matrix predicted, while only being able to score a medium number of cans, our robot was both fast and reliable. During the competition, many of the robots failed completely and could not compete, showing that these two attributes (speed and reliability) were among the most important to have. Our decision to go with a simple robot over a complex stacking robot paid off when our robot performed as expected, not failing for nearly the entire competition. With our method of pushing with a hard coded path being among the fastest, we were also ready to go on the offensive and steal cans as soon as the competition allowed it.

The one true requirement that was given to us for the competition was to produce a point scoring robot. We not only completed this goal, but scored among the most points of any robot during the entire competition. Late in the competition however, an inherent flaw came out in the design of our robot that was not previously thought of as an important error. Our robot's design, while very reliable at performing in the competition, had exposed wires. At the end of one of the matches, our robots wires got accidentally disconnected due to being caught on the opposing team's robot. Our team had to scramble to find what wires went where at the last minute because we had not made a reference or manual. After this incident, our robot quickly lost two matches, showing the significance that this flaw actually carried. While this error was relatively significant, our robot still performed very well, and as a whole was a great success.

### **Budget and justification and Bill of Materials**

Our budget consisted of one hundred fifty dollars, which is refunded after the competition and whatever funds we were willing to put up for the robot ourselves

Item	Justification/Supplier	Cost
Base Structure (wood)	Used for the core structure of the robot	\$30
PolyBot board	Pre-Built, rented for duration of project	\$20
Motors, Mounts and Wheels	Purchased from pololu.com	\$65
Line Detecting sensors	Purchased from Sparkfun.com	6 x \$3= \$18
Other	Various other Components and expenses (wires, screws, glue, ...)	\$20
Total of Costs	-	\$153

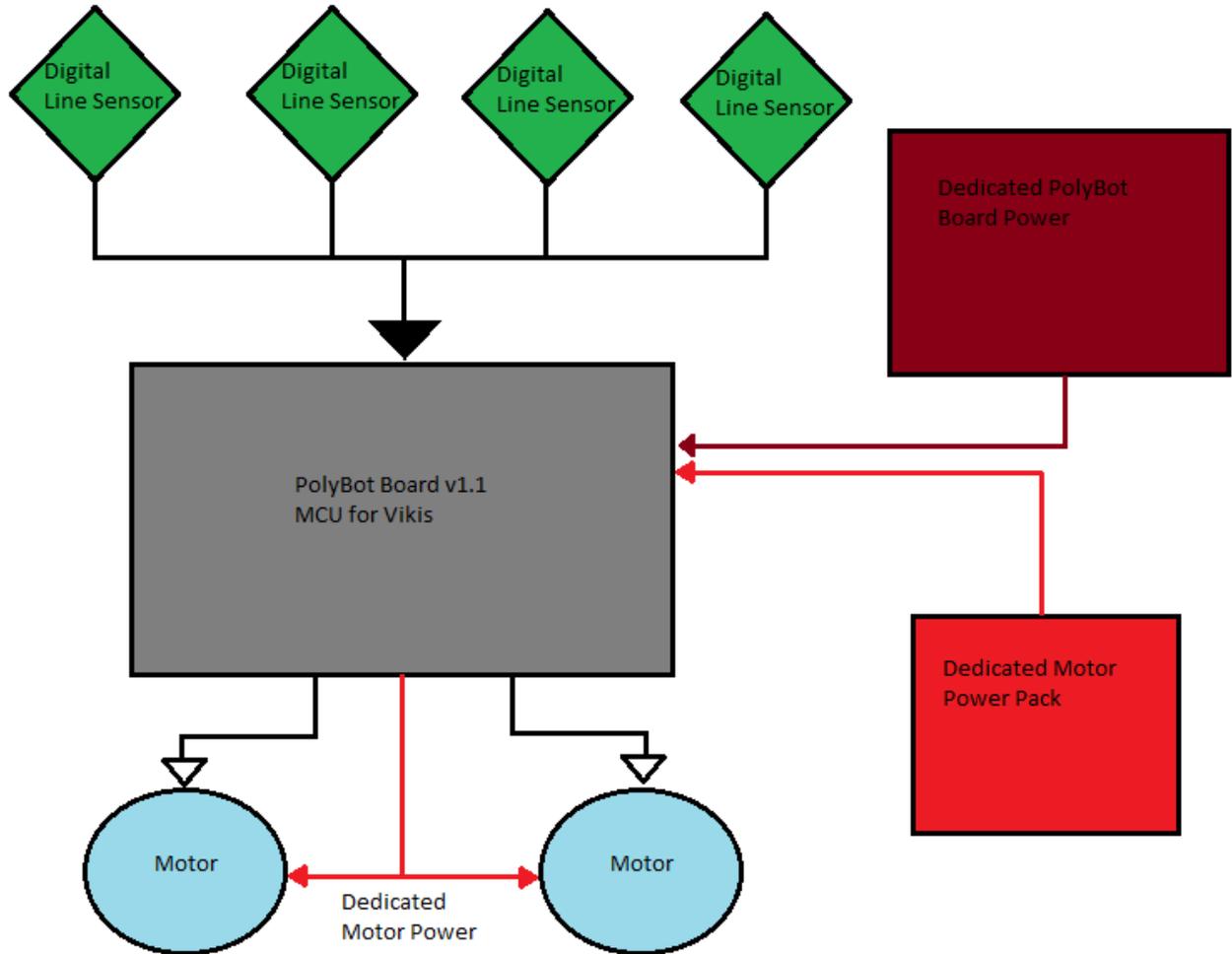
### **References**

PolyBot board - <http://users.csc.calpoly.edu/~jseng/PolyBotBoard.html>

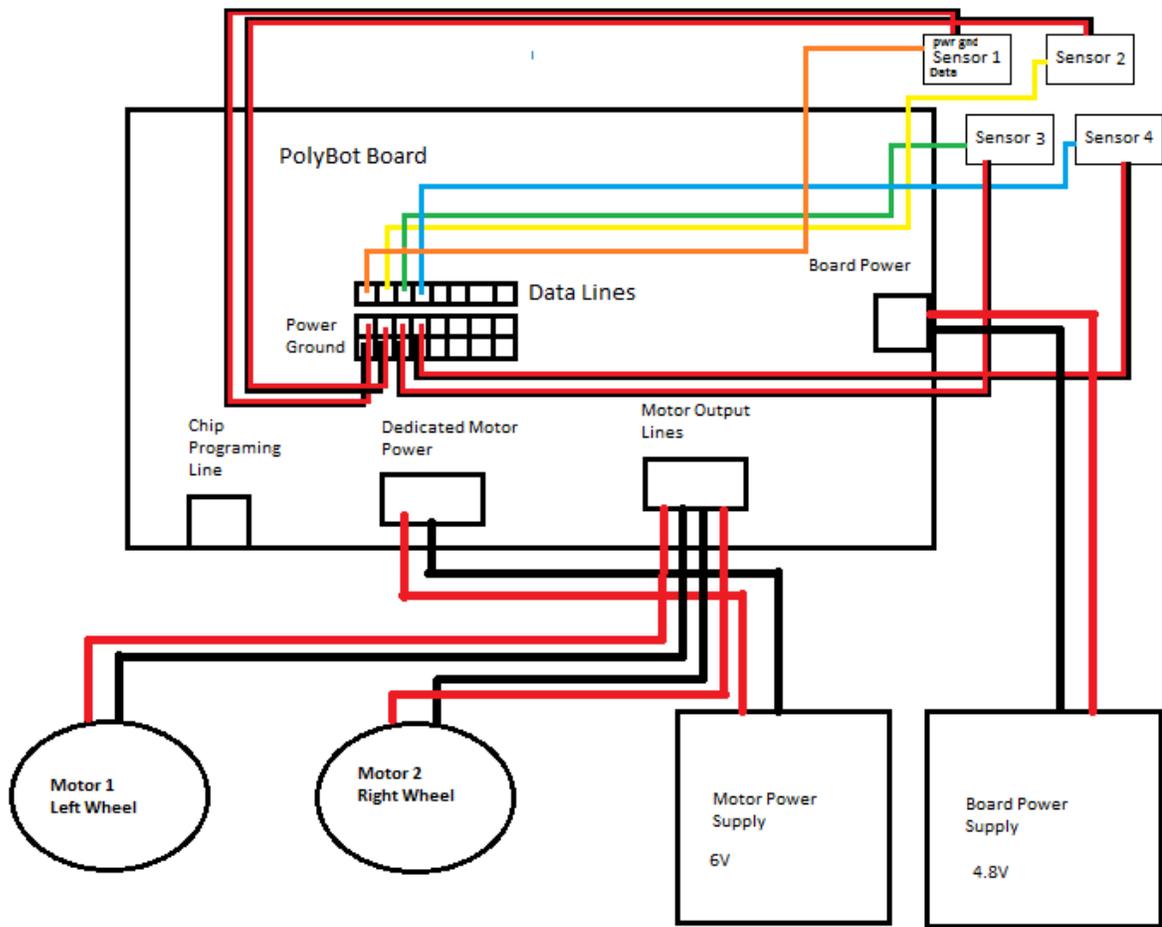
Line Sensors - <http://www.sparkfun.com/products/9454>

Motors - <http://www.pololu.com/catalog/product/1574>

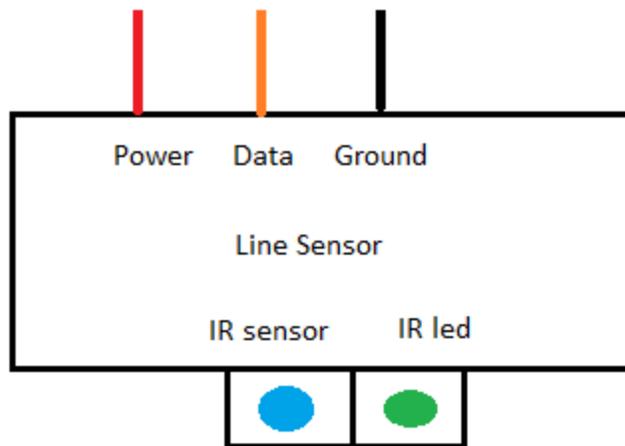
*Appendix A Diagrams*



*Figure 3. Hardware Block Diagram for Vikis*



**Figure 4. PolyBot Pin Out Diagram**



**Figure 5. Line Detecting Sensor**

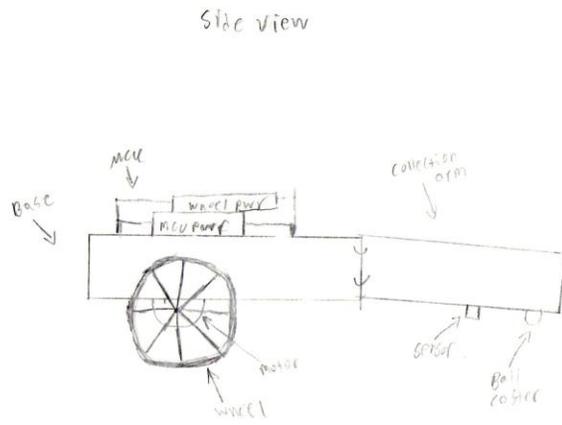
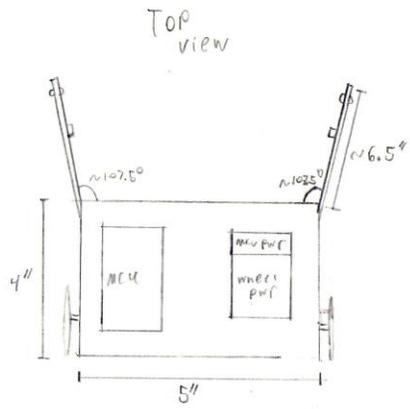


Image 1. Vikis Schematic



*Image 2. Top down view of Vikis*



*Image 3. Side view of Vikis*

## ***Appendix B Function Declarations, Descriptions & Code***

*void one\_row();*

The *one\_row()* function calls various functions below, like orientate and deposit, to collect one row of friendly cans, deposit them in the goal zone, then reorientated Vikis to begin collect of another row.

*void deposit();*

The *deposit()* function orientates Vikis with the friendly goal zone and deposits all cans in his collection area, then turns around and orientates at the edge of the goal zone facing the play area.

*void wall();*

The *wall()* function handles turning Vikis when he is close to a wall, ensuring that he does not get stuck on that wall and does not lose cans.

*void wall2();*

The *wall2()* function also handles turning Vikis when he is close to a wall, ensuring that he does not get stuck on that wall and does not lose cans. This one differs from *wall()* because it is specifically designed for the situation when Vikis is pushing lots of cans.

*void forward(int speed);*

The *forward()* function takes in a speed value between -100 (full reverse) and 100 (full forward) and set Vikis's movement speed to this value.

*void reverse(int speed, int interval);*

The *reverse()* function takes in a speed value and an time interval and send Vikis moving in that direction for the given period of time.

*void pivot\_right(int speed, int interval);*

The *pivot\_right()* function takes in a speed and a time interval and turns Vikis in place at the speed for that duration of time. This turn is a pivot because only the left wheel moves during the turn.

*void pivot\_left(int speed, int interval);*

The *pivot\_left()* function takes in a speed and a time interval and turns Vikis in place at the speed for that duration of time. This turn is a pivot because only the right wheel moves during the turn.

*void swing\_right(int speed, int interval);*

The *swing\_right()* function takes in a speed and a time interval and turn Vikis to the right at that speed for that duration. This turn is a swing because both wheels are moving during the turn. The wheels spin at different speeds to accomplish a swinging motion.

*void swing\_left(int speed, int interval);*

The *swing\_left()* function takes in a speed and a time interval and turn Vikis to the left at that speed for that duration. This turn is a swing because both wheels are moving during the turn. The wheels spin at different speeds to accomplish a swinging motion.

*void swing\_right\_r(int speed, int interval);*

The *swing\_right\_r()* function takes in a speed and a time interval and turn Vikis to the right in reverse at that speed for that duration. This turn is a swing because both wheels are moving during the turn. The wheels spin at different speeds to accomplish a swinging motion.

*void swing\_left\_r(int speed, int interval);*

The *swing\_left\_r()* function takes in a speed and a time interval and turn Vikis to the left in reverse at that speed for that duration. This turn is a swing because both wheels are moving during the turn. The wheels spin at different speeds to accomplish a swinging motion.

*void stop();*

The *stop()* function sends the commands to stop all movement of Vikis.

*void orientate();*

The *orientate()* function takes in data from the line detecting sensors and spins the corresponding wheels forward to orientate the sensors on Vikis to be over the line. This causes Vikis to be parallel with the line and therefore know his position relative to the goal zone or other areas of the course, based on timing.

*void orientate\_r();*

The *orientate\_r()* function takes in data from the line detecting sensors and spins the corresponding wheels in reverse to orientate the sensors on Vikis to be over the line. This causes Vikis to be parallel with the line and therefore know his position relative to the goal zone or other areas of the course, based on timing.

```
#include "globals.h"
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
void one_row();
void deposit();
void wall();
void forward(int speed);
void reverse(int speed ,int interval);
void pivot_right(int speed, int interval);
void pivot_left(int speed, int interval);
void swing_right(int speed, int interval);
void swing_left(int speed, int interval);
void swing_right_r(int speed, int interval);
void swing_left_r(int speed, int interval);
void stop();
void orientate();
void wall2();
void orientate_r();
```

```
int main(void) {
    initialize();
    int prog=0, bool_=1;
    int loop;
    led_on();
    delay_ms(200);
    led_off();
    delay_ms(200);
    led_on();
    delay_ms(200);
    led_off();
    delay_ms(200);
    /*this is the start up msg*/
    print_string("Start Up");
    next_line();
    print_string("Select a Prog");
    delay_ms(1500);

    clear_screen();
    /*this while loop is where you hold down the button
    * to select a program 0-x
    */
    while(bool_){
        print_string("prog");
```

```

    print_int(prog);
    delay_ms(500);
    if(get_sw1()==257)
    {
        prog++;
        //if(prog>10);
        // prog=0;
    }
    else
        bool_=0;
    clear_screen();
}
/*this just prints the program that was selected*/
clear_screen();
print_string("Ready" );
next_line();
print_string("Prog " );
print_int(prog);
print_string(" Selected");
delay_ms(2000);
clear_screen();
/*here is where the program that was chosen start executing
thus far program 0 is the one we are using so any changes should be made in here
or in the functions called in here*/
if(prog==1)
{
    orientate();
    pivot_left(50, 600);
    pivot_right(50, 500);
    orientate();
    pivot_left(75, 600);
    pivot_right(75, 500);
    orientate();
    pivot_left(75, 700);
    pivot_right(75, 600);
    forward(100);
    delay_ms(2000);
    pivot_left(100, 800);
    reverse(0, 100);
    deposit();
}
/* WHERE USING THIS ONE*/
else if(prog==0)
{
    print_string("Collection");
    orientate();// this is the first orientation call that assures the robot starts on the line
    one_row(1);// this is the pivot 90 and then orientate on the 3 lines with the cans
    forward(120);
}

```

```
delay_ms(2800);
pivot_right(120, 1000);
orientate();
deposit();
```

```
forward(100); // move forward for .45 sec to recollect cans and offset to the left some
delay_ms(1000); //<---- change this one to change duration dont change the forward() command
swing_left(100, 600); //start by backing up and swinging the front to the left
orientate();
```

```
    pivot_left(50, 600);
    pivot_right(50, 500);
    orientate();
    pivot_left(75, 600);
    pivot_right(75, 500);
    orientate();
    pivot_left(75, 700);
    pivot_right(75, 600);
    forward(100);
    delay_ms(2500);
    pivot_left(100, 800);
    reverse(0, 100);
    deposit();
    forward(100);
    delay_ms(700);
```

```
    swing_right(100, 300);
    /// attack code starts here
    clear_screen();
```

```
    print_string("DONE");
reverse(0, 1000);
    //stop();
    delay_ms(15000);
    forward(100);
    delay_ms(2250);
    pivot_left(100, 300); //<-----
    forward(100);
    delay_ms(2250);
    pivot_right(100, 600);
    forward(100);
    delay_ms(1500);

    pivot_right(100, 300);
    forward(100);
    delay_ms(500);
    pivot_right(100, 500);
    forward(100);
```

```

    delay_ms(5000);
    reverse(0,100);
    pivot_right(100,300);
    deposit();
    pivot_left(100, 200);
    forward(100);
    delay_ms(3000);
    pivot_right(100, 300);
    forward(100);
    delay_ms(5000);
    swing_left_r(100,500);
    forward(100);
    delay_ms(200);
    swing_left_r(100, 500);
    forward(100);
    delay_ms(200);
    swing_left(100, 300);
    forward(200);
    delay_ms(100);
    swing_left(100, 200);//<-----
    forward(100);
    delay_ms(2000);
    swing_left(100, 600);
    forward(100);
    delay_ms(2000);
    swing_left(100, 300);
    forward(100);
    delay_ms(5000);
    //deposit();
    forward(0);
    pivot_right(100, 500);//<---- new as of this morning
    deposit();
    pivot_right(100, 500);
    forward(100);
    delay_ms(6000);

    /* pivot_left(100, 200);
    forward(100);
    delay_ms(3000);
    orientate();*/
}
else if(prog==2)
{
    orientate();
    pivot_right(100, 300);
    forward(100);
    delay_ms(700);

```

```
pivot_left(100, 300);//<----
forward(100);
delay_ms(3000);
pivot_right(100, 600);
forward(100);
delay_ms(1500);

pivot_right(100, 300);
forward(100);
delay_ms(500);
pivot_right(100, 500);
forward(100);
delay_ms(5000);
reverse(0,100);
pivot_right(100,300);
deposit();
pivot_left(100, 200);
forward(100);
delay_ms(3000);
pivot_right(100, 300);
forward(100);
delay_ms(5000);
swing_left_r(100,500);
forward(100);
delay_ms(200);
swing_left_r(100, 500);
forward(100);
delay_ms(200);
swing_left(100, 300);
forward(200);
delay_ms(100);
swing_left(100, 200);//<-----
forward(100);
delay_ms(2000);
swing_left(100, 600);
forward(100);
delay_ms(2000);
swing_left(100, 300);
forward(100);
delay_ms(5000);
//deposit();
forward(0);
pivot_right(100, 500);//<---- new as of this morning
deposit();
pivot_right(100, 500);
forward(100);
delay_ms(6000);
```

```

        /* pivot_left(100, 200);
        forward(100);
        delay_ms(3000);
        orientate();*/

    }
    return 0;
}
/* the sequence for deposit should be...
move forward pushing the cans into the goal zone
then move back so when it pivots to reorientate itself the cans
clear the front sweepers, then pivot times moving forward between each one
so the robot moves onto the next row to collect
Finally it calls orientate in reverse to ensure it is in the
same position for the start of one row collection, except its one row to
the left from the last time
*/
void deposit()
{
    forward(100); //these two lines tell it to move forward for 1.9sec
    delay_ms(3000);
    reverse(100, 1000); //move back for 1.4 sec
    pivot_right(100, 1000); //pivot left for .7sec
    forward(100); //these two are move forward for 1sec
    delay_ms(100);
    swing_left_r(-100, 1400); //swing right in reverse for 1.5 sec
    orientate_r(); // orientate in reverse stopping on the line
    //pivot_left(100, 900);
    //orientate();

}
/*collect one row*/
void one_row(int loop)
{
    orientate();
    forward(100);
    delay_ms(600); //these two lines tell it to move forward until just before it will hit the wall
    if(loop==0){
        wall(); // basically the code to turn around and offset one row to the left
    }
    else if(loop==1)
    {
        wall2(); // same as wall
    }
    // orientate();
    // pivot_right(100, 700);
    // forward(100, 1000);
    // pivot_right(100, 1200);

```

```

// forward(100);
}
/* wall and wall 2 are the same
there are two of them (and maybe more if needed)
so that the times can be tweaked for each row that needs to
be collected
*/
void wall()
{
  swing_left_r(-100, 1000); //start by backing up and swinging the front to the left
  forward(100); // move forward for .45 sec to recollect cans and offset to the left some
  delay_ms(800); //<---- change this one to change duration dont change the forward() command
  pivot_right(100, 750); //back up and swing the robots nose to the left
  forward(100); //move forward again (same reasons)
  delay_ms(500); //<----- change this one
  pivot_right(100, 750); //swing again (should be almost if not past 90 degrees now)
  forward(100); //forward
  delay_ms(700); //<----- change this one
  pivot_right(100, 1000); //last swing should be almost 180 now
  forward(100); //last forward
  delay_ms(1000); //<----- change this one
  //its ok if its not a full 180 because right after wall is called orientate() is called
  //this should take care of any small discrepancy at the end of the turn around.

}
/* this is the code to make it turn 90 degrees then orientate
on the three virtline
*/
void wall2()
{
  swing_left_r(-100, 500);
  forward(100);
  delay_ms(300);

  swing_left_r(-100, 600);
  forward(100);
  delay_ms(400);
  pivot_right(100, 400);
  orientate();
  orientate();
  orientate();

}
//-----
//DONT CHANGE ANYTHING BELOW THIS LINE!
//-----
void forward(int speed)

```

```

{
  int speed2=speed;
  int right=set_motor_power(0, speed-5);
  int left=set_motor_power(1, speed);
  delay_ms(100);
}
void stop()
{
  set_motor_power(0,0);
  set_motor_power(1,0);
}
void reverse(int speed ,int interval)
{
  set_motor_power(1, -speed);
  set_motor_power(0, -speed);
  delay_ms(interval);
  set_motor_power(0,0);
  set_motor_power(1,0);
}
void pivot_right(int speed, int interval)
{
  set_motor_power(0, speed);
  set_motor_power(1, -speed);
  delay_ms(interval);
  set_motor_power(0,0);
  set_motor_power(1,0);
}
void pivot_left(int speed, int interval)
{
  set_motor_power(0, -speed);
  set_motor_power(1, speed);
  delay_ms(interval);
  set_motor_power(0,0);
  set_motor_power(1,0);
}
void swing_right(int speed, int interval)
{
  set_motor_power(1, -speed);
  delay_ms(interval);
  set_motor_power(0,0);
  set_motor_power(1,0);
}
void swing_left(int speed, int interval)
{
  set_motor_power(0, -speed);
  delay_ms(interval);
}

```

```

    set_motor_power(0,0);
    set_motor_power(1,0);
}
void swing_right_r(int speed, int interval)
{
    set_motor_power(0, -speed+75);
    set_motor_power(1, -speed);
    delay_ms(interval);
    set_motor_power(0,0);
    set_motor_power(1,0);
}
void swing_left_r(int speed, int interval)
{
    set_motor_power(1, speed);
    set_motor_power(0, speed+75);
    delay_ms(interval);
    set_motor_power(0,0);
    set_motor_power(1,0);
}
void orientate()
{
//sensor left=0 right=1
// second sensor left 2, right 3
//wheels left=1 right=0

int bool1=0;
int bool2=0;
int count=0;
set_motor_power(0,100);
set_motor_power(1,100);
delay_ms(500);
while(1){

count++;
if(digital(1) || digital(3))
{
    set_motor_power(0,0);
    delay_ms(100);
    //print_string("left det");
    bool1++;
}
if(digital(0) || digital(2))
{
    set_motor_power(1,0);
    delay_ms(100);
    //print_string("right det");
    bool2++;
}
}
}

```

```

}
if(bool1>=1 && bool2>=1)
    break;
else if(bool1 && !bool2 && count>100)
    break;
else if(!bool1 && bool2 && count>100)
    break;
else if(count>300)
    break;

        clear_screen();
        print_int(count);
}
set_motor_power(0,0);
set_motor_power(1,0);
delay_ms(100);
clear_screen();
print_string("done");
}

```

```

void orientate_r()
{
//sensor left=0 right=1
//wheels left=1 right=0
int count=0;
int bool1=0;
int bool2=0;
set_motor_power(0, -100);
set_motor_power(1, -100);
delay_ms(100);
while(1){

count++;
if(digital(1)==1)
{
set_motor_power(0,0);
delay_ms(100);
print_string("left det");
bool1++;
}
if(digital(0)==1)
{
set_motor_power(1,0);
delay_ms(100);
print_string("right det");
bool2++;
}
}
}

```

```
if(bool1>=1 && bool2>=1)
    break;
else if(bool1 && !bool2 && count>250)
    break;
else if(!bool1 && bool2 && count>250)
    break;
else if(count>250)
    break;
    clear_screen();
        print_int(count);

}
    set_motor_power(0,0);
    set_motor_power(1,0);
    delay_ms(100);
}
```