CPE Senior Project Final Report:
Roborodentia 2012 - Team Street Sweeper

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

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Science

by

Jason Miller
David Pascale

June, 2012

Table of Contents

Executive Summary

The following report describes the process by which we developed a fully autonomous robot in order to submit it to Cal Poly's 17th Annual Roborodentia competition.

With regard to the development of such a robot, there was a significant amount of work put in, particularly in the design, to develop a robot that would be capable of contending with robots developed by a collection of some of the brightest minds at Cal Poly.

The first step in our process was to devise a strategy that could effectively meet the demands of the competition. We decided upon a plan in which we would sneak into the enemy robot's goal area and steal their cans, as well as knock down any accumulated stacks.

Next, we developed a robot that would be durable, quick, and inexpensive. The robot consisted mostly of wood, which allowed for easy body frame manipulation, as well as component mounting. This allowed us to be flexible with our manufacturing, and make real-time decisions in the process of construction and testing.

Most of all, the software defined whether or not our plan would work. Our software utilized two simple line sensors in order to track where the robot was headed and stay on course. Keeping a simple design allowed us to make incremental changes, yet keep the coding simple enough to efficiently detect errors and correct them.

This simple design methodology was underestimated by our opponents. While their robots struggled from overly-complicated designs, we managed to sneak in and steal the competition, taking second place overall.

While there were certain things that we could have certainly done better, this was a great learning experience in terms of understanding Arduino and even robots in general. Considering that this was the first robot ever assembled by either member of our group, we feel as if we were very successful in our planning and execution.

Competition Overview

For Roborodentia XVII, there was a clear set of rules that were very influential in the formation of a particular strategy for the competition. The complete competition rules are attached in Appendix A of this report.

The key aspects of the rules that were most important to consider were those that set size requirements, those that dictated how the robot operated, and those that stated what the robot could or could not have on it.

For this particular competition, the robot was to be completely autonomous and self-contained, as well as within the size footprint of 12” wide X 12” long at the beginning of the match, with the ability to expand to 14” wide X 14” long during the match. Height, however, was unrestricted. Additionally, the robot could not be designed to do damage to other robots, nor could it have wireless transmitters/receivers.

As for the competition itself, the scoring methodology for the cans was also very important, since the functionality of the robot directly affected the ability to score cans in certain ways. For example, if the robot constructed has no stacking ability, then it limits the robot’s scoring potential to one point for returning its own cans to the goal area and two points for returning its opponent’s cans. However, the stacking aspect allowed for an extra two points per consecutive can stacked. This means that, had a robot stacked its own cans, the bottom can would be one point; the next up would be three points, then five points, and so on. The same concept occurred with opponent cans, with the scoring proceeding at a sequence of two, then four, then six points, and so on.

The course specifications are provided in Figure 1.

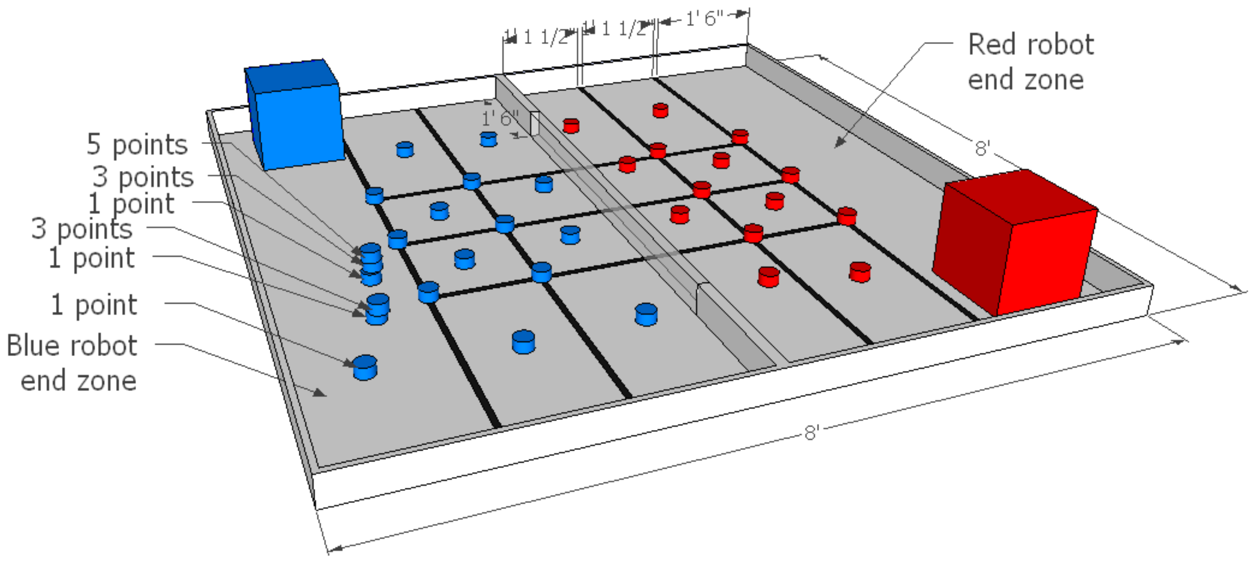

Figure 1: Roborodentia Course Layout

Project Overview

When considering robot design, first we decided it would be best to develop the strategy based on what we thought other teams would be doing, given the competition rules. Since the stacking method was very appealing due to the amount of points that could be tallied quickly, we decided to prepare to steal stacked piles from our opponent, or simply knock the stacks over while on their side. We figured that most teams would be so focused with collecting cans for stacking that they would not expect a team to keep such a simple tactic as stealing cans from their opponent.

So, given this decision, we started to generate ideas that would help this method thrive. The robot would have to be fast, yet durable, and also have the ability to move multiple cans at one time. Additionally, it would need some sort of ability of detecting where it was and utilizing competition field markers to follow a path back to the goal area. Noting these qualities, we then began designing our robot.

Design

Design Ideas

We originally came up with three ideas for possible robot designs. We decided that our robot should be a so-called “dumb agent”, meaning it would not react to percepts, but rather just follow a pre-programmed path. This made the robot much simpler and cheaper to build because it did not require any extra sensors. The only sensor required were line sensors so it could follow a straight line, which was very important in successfully following a path.

The first idea was similar to our final design, but instead only had one raise-able arm on one side that lifted up when the robot moved forward. When the robot began to move backwards the arm lowered and swept any cans backwards with it, and into the end zone. The robot then moved over and repeated until all the cans had been collected. It was similar to our final design in that it did not worry about attempting to stack cans, but was more about quickly collecting as many as possible and stealing the opponent’s cans.

The second design focused on stacking cans. This robot would have a claw in the front. When the claw touches a can it would squeeze it and flip it backwards into a tube with a door on the bottom. When the robot collects and stacks a certain number of cans it would return to the end zone, open the door on the bottom of the tube and raise the tube, leaving the stacked cans in the end zone. The robot would then continue searching for more cans. This design was developed in case we wanted to attempt a robot that had stacking capabilities if we simply could not match the scoring output of the other robots through our stealing tactics.

Our last and final design idea was a modification of the first design. Instead of having a robot with arm that had to return to the start quite often, the robot would have two arms that collected as many cans as possible before returning. The two arms would attempt to trap the cans within its arms and could lift the arms up to drop the cans off at our end zone. Figure 2 shows our early drawings of our final design. In the end, we did not use the multi-directional wheel for stability.

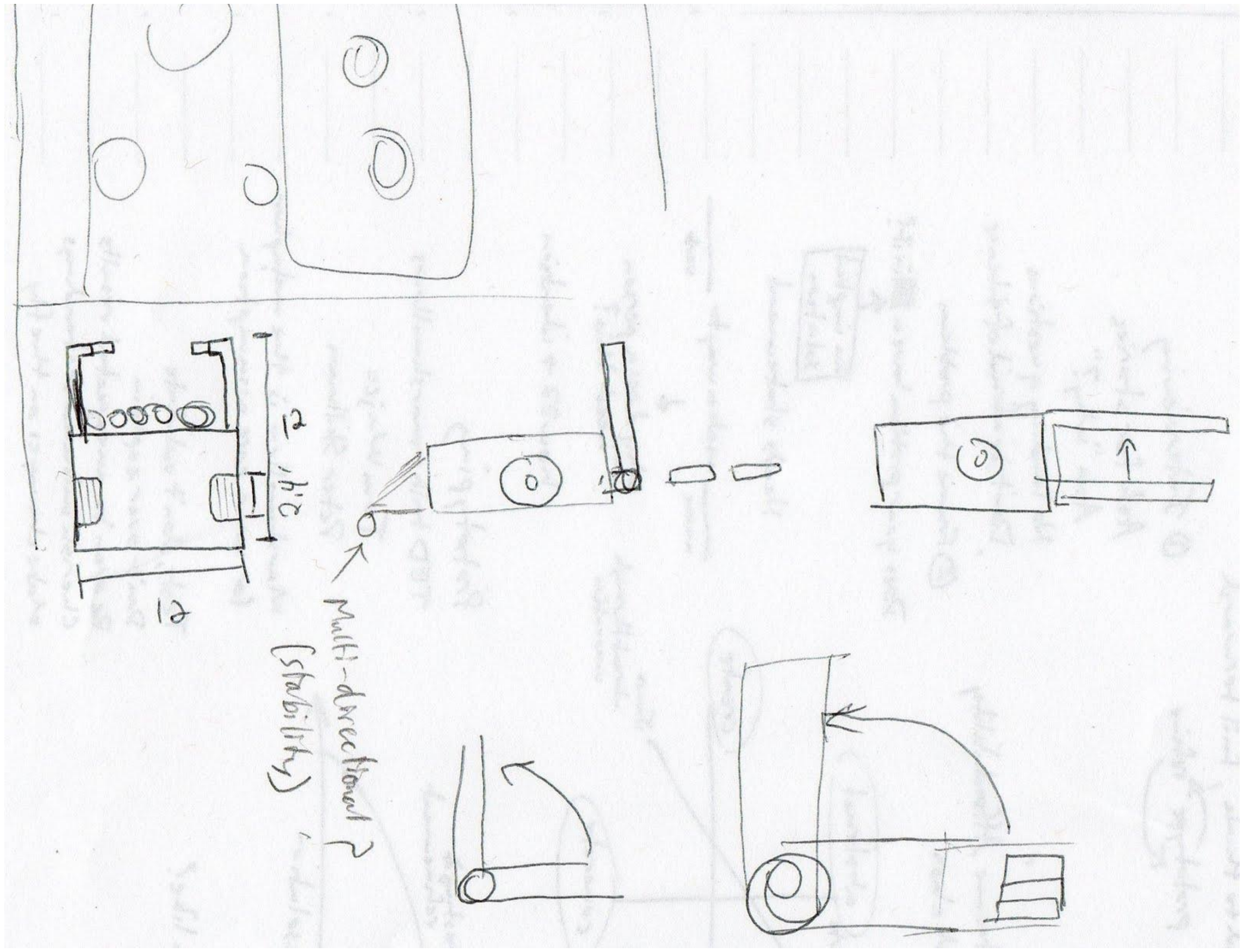

Figure 2: Early drawings for final design

Additionally, it can be seen in Figure 2 that we were considering different ways to potentially lift the arms, with each method having its own merits and limitations.

Final Design

We had decided to keep to our initial idea of having a basic thief strategy, where the robot is light and fast, yet durable. Keeping this in mind, we decided upon having two wheels so that there was minimal weight added to the robot while having more consistent turning. We also decided that poplar wood would be a good option for the body frame since it was light, cheap, tough, and easy to shape. Additionally, it was decided that having arms of galvanized steel would allow us the strength capabilities to move groups of cans without fracture, while still keeping overall robot weight to a minimum.

After formation of the frame, the motors and wheels were mounted, as well as the servos for arm operation. Since there were only two wheels, the robot’s lack of balance caused it to teeter back and forth, so by focusing all the weight of the battery towards the front of the robot, we were able to shift the robot’s center of gravity to its front, and then create a balancing post from excess wood. This post protruded from the bottom of the robot, and kept the main body parallel with the ground. We then covered the bottom of the post with moleskin to reduce rolling friction.

We then attached the arms by duct-taping the steel pieces together and covering the bottom surface of the arm in moleskin to reduce friction with the ground.

Another feature of the robot was that the main body had two pieces of poplar enclosing all necessary electronics on the top and bottom, so as to prevent any damage during the course of the robot’s usage. The final robot, the Street Sweeper, is displayed in Figures 3 and 4, with Figure 4 showing the interior wiring exposed.

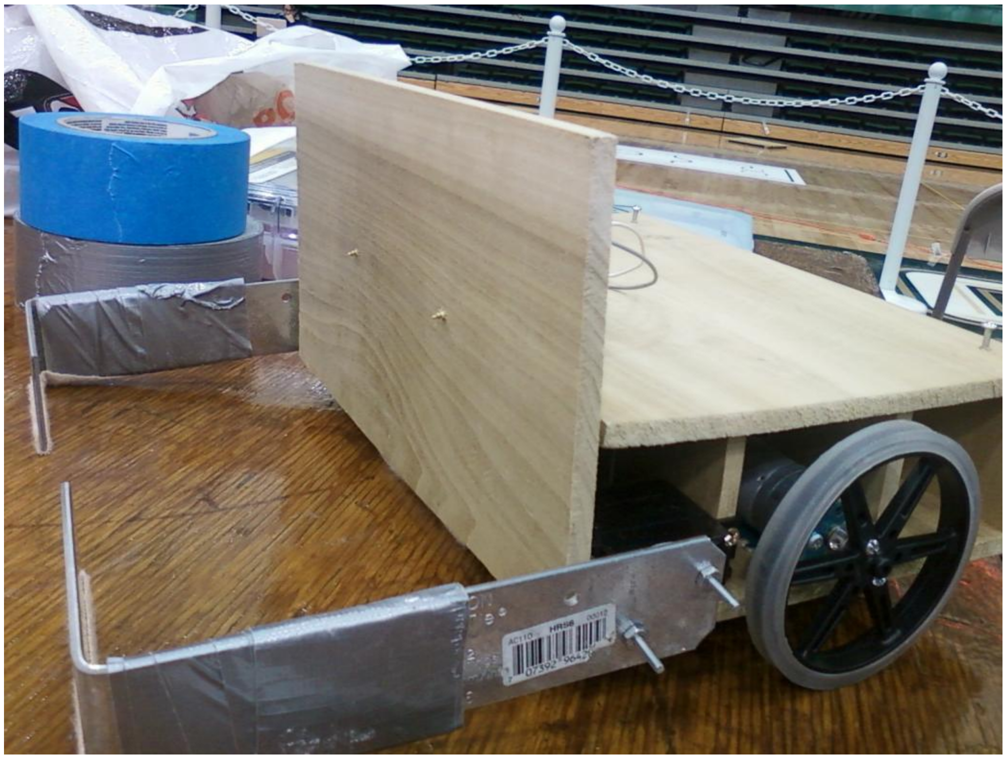

Figure 3: Finished robot

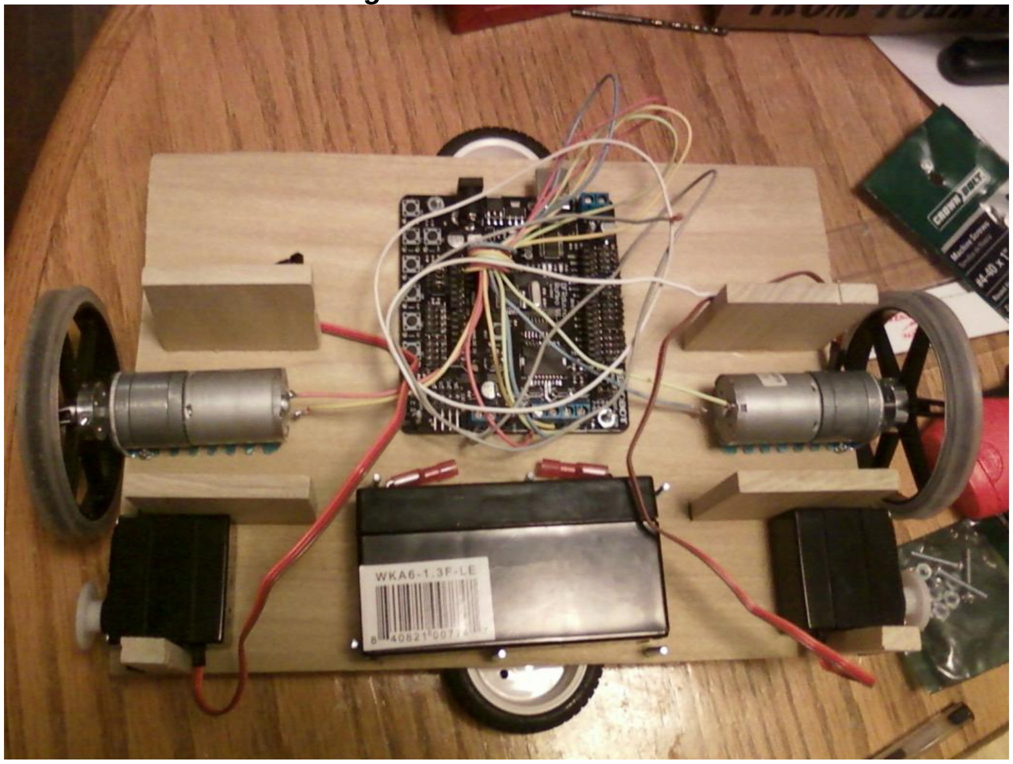

Figure 4: Servos and motors attached to Romeo controller

In addition to having a simple strategy, the spartan characteristics of the robot allowed us to have a fairly simple electronic set-up. Not pictured in Figure 3 is the addition of line sensors, which were very key in the robot's functioning. These sensors were positioned in the empty space between the battery and the microcontroller such that there was a small enough gap between them so that they could detect the same line, but there was a large enough gap such that one could still be detecting a line while the other lost contact. The final bill of materials is as follows in Table 1:

Component Wiring

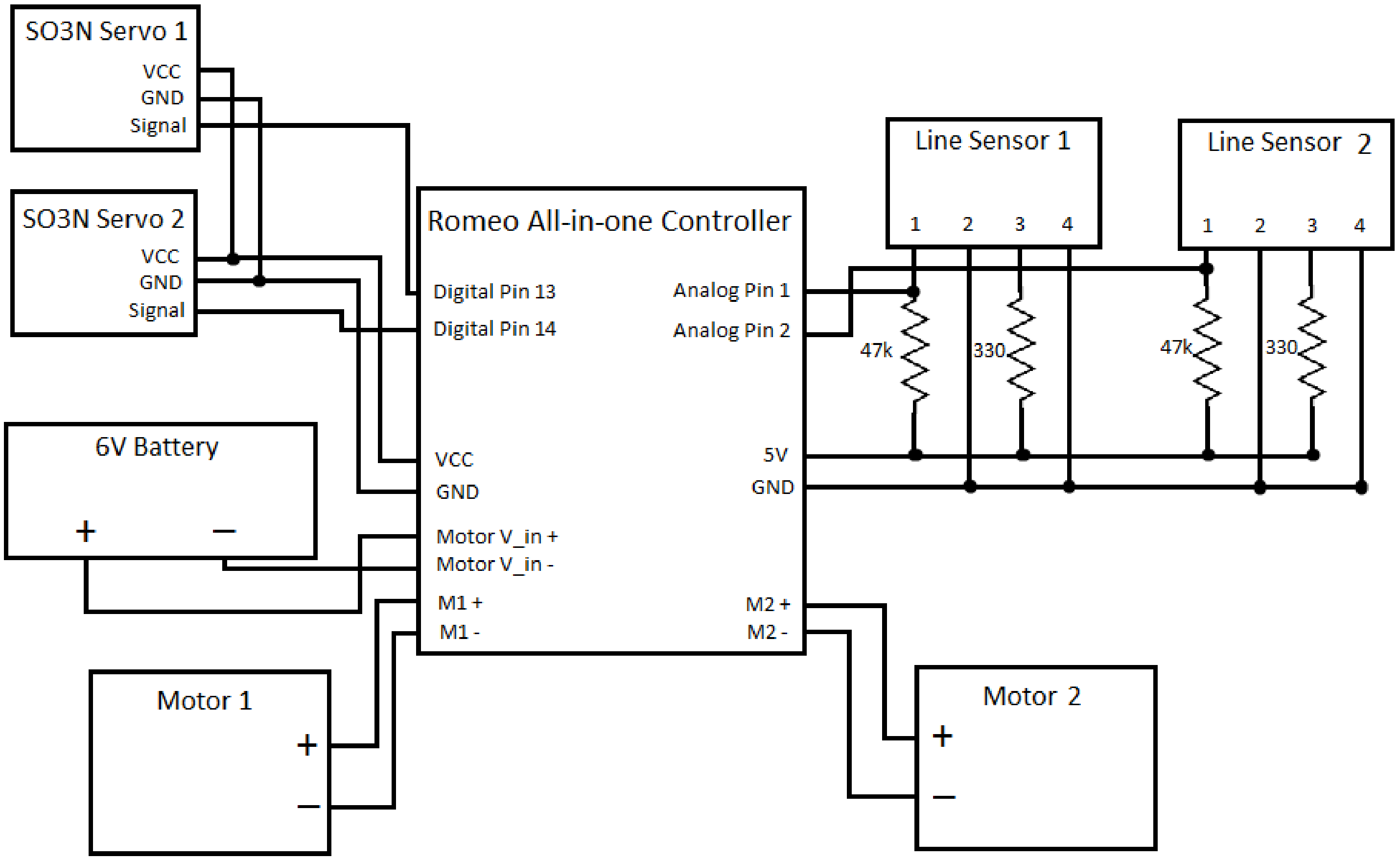

Figure 5: Overall System Block Diagram

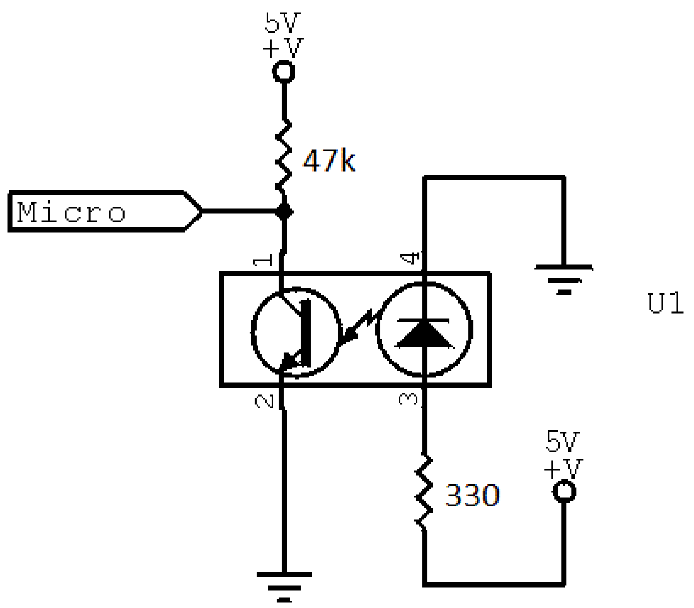

Figure 6: Circuit diagram of QRD1114 Line Sensor. Source: http://www.hvwtech.com/products_view.asp?ProductID=97

Figure 5 shows the overall system block diagram for Street Sweeper. The Romeo All-in-one Controller (http://www.dfrobot.com/index.php?route=product/product&product_id=56) contains an ATMega 328 microprocessor and controls every component of the system. The motors are connected to special motor pins M1 and M2 which can control the speed and direction of each motor. The motors are powered by a 6V battery which is connected to the V_in terminals. The line sensors are connected to analog pins 1 and 2. Each sensor requires pull up resistors on the collector of the transistor as well as on the positive end of the LED, both contained within each line sensor [3]. Figure 6 shows the circuit diagram of a QRD1114 line sensor. The servos are connected to digital pin 13 and 14 and can be sent values corresponding to degrees for which the servos can raise or lower. The servos are powered by the 5V VCC pin.

Bill of Materials

Product	Quantity	Price
Romeo All-In-One Controller	1	$44.50
GWS S03N Standard Servo	2	$23.90
Pololu Universal Aluminum Mounting Hub Pair	1	$6.95
Tamiya 70111 Sports Tire Set (2 tires)	1	$6.50
47:1 Metal Gearmotor 25Dx52L mm	2	$39.90
Pololu Wheel 80x10mm Pair - Black	1	$9.25
High-Traction Stick Tire (one tire)	2	$6.50
Polulu 25D mm Metal Gearmotor Bracket Pair	1	$7.45
WKA6-1.3F 6V Battery and charger	1	$34.46
½"x4x4 Poplar Board	2	$11.84
Simpson Strong-Tie Angle	2	$5.14
Simpson Strong-Tie 6-in. Heavy Duty Strap	2	$3.76
TOTAL COST		**$200.15**

Table 1 - Bill of Materials

Software Algorithms

The software was written using the Arduino language and the Arduino IDE. The only external library required was the Servo library. The full source code can be seen in Appendix B.

Robot Movement

For the robot to move, you must specify the speed of each motor and in what direction they are turning. The four lower level functions to do this are turn_R, turn_L, advance, and back_off, shown below.

```
void advance(char a,char b)           //Move forward
{
  analogWrite (E1,a);   //PWM Speed Control
  digitalWrite(M1,HIGH);
  analogWrite (E2,b);
  digitalWrite(M2,HIGH);
}
void back_off (char a,char b)            //Move backward
{
  analogWrite (E1,a);
  digitalWrite(M1,LOW);
  analogWrite (E2,b);
  digitalWrite(M2,LOW);
}
void turn_R (char a,char b)              //Turn Left
{
  analogWrite (E1,a);
  digitalWrite(M1,LOW);
  analogWrite (E2,b);
  digitalWrite(M2,HIGH);
}
void turn_L (char a,char b)              //Turn Right
{
  analogWrite (E1,a);
  digitalWrite(M1,HIGH);
  analogWrite (E2,b);
  digitalWrite(M2,LOW);
}
```

Each of these functions takes in a character representing the speed of each motor (‘a’ to control M1 and ‘b’ to control M2). Advance() will set both motors to the values of a and b and set the direction of both motors to forward. Turn_R() and turn_L() are similar, except they set one of the motors to turn in the opposite direction to make it turn. Back_off() is the same as advance() except it sets the motors to rotate in the opposite direction as advance().

Because most of the time we used the maximum values for the speed of each motor (255), we created 4 more functions that add a higher level of abstraction, making it easier to create paths for the robot to follow, without having to worry about speeds and delays. These functions, turnRight(), turnLeft(), moveForward(), and moveBackward() are shown below.

```
/*turn right at full speed for <time> milliseconds*/
void turnRight(int time)
{
  turn_R(255, 255);
  delay(time);
  stop();
}

/*turn left at full speed for <time> milliseconds*/
void turnLeft(int time)
{
  turn_L(255, 255);
  delay(time);
  stop();
}

/*advance at full speed for <time> milliseconds*/
void moveForward(int time)
{
  advance(255, 255);
  delay(time);
  stop();
}

/*move backwards at full speed for <time> milliseconds*/
void moveBackward(int time)
{
  back_off(255, 255);
  delay(time);
  stop();
}
```

The functionality of these is the same, except it assumes full speed for the motors and instead takes in a delay time. This delay time tells the robot how long it will carry out a certain movement. We also created a function to turn the robot 180 degrees, as this was a common motion for our robot:

```
void turn_180()
{
  /*Turn 180 degrees*/
  turn_R(255,255);
  delay(1500);
  stop();
}
```

Line Sensing

In addition to these functions are followPath() and followLine(). FollowPath() is the main function that contains all the movement commands and is shown in Appendix B. FollowLine(), shown below, is a function that allows the robot to follow the black line from one side of the playing field to the other.

```
/*Function that follows a black line from one side to the other*/
void followLine()
{
  int i;
  for (i=0; i<7000; i++)
  {
        left = analogRead(analogPin);
        right = analogRead(analogPin2);

        if (left < 400 && right > 400)
        {
        //Serial.println("Veering left");
        advance(220, 255); //Decrease right motor speed
        lastDetected = 1;
        }
        else if (right < 400 && left > 400)
        {
        //Serial.println("Veering right");
        advance(255, 220); //Decrease left motor speed
        lastDetected = 2;
        }
        if ((left < 400 && right < 400) && lastDetected == 1)
        {
        //Serial.println("Veering left, lastDetected");
        advance(220,255);
        }
        else if ((left < 400 && right < 400) && lastDetected == 2)
        {
        //Serial.println("Veering right, lastDetected");
        advance(255,220);
        }
        else
        {
        advance(255,255);
        }
        delay(1);
  }
}
```

FollowLine() works by reading from the line sensors once every millisecond. If one of the sensors detects white, then it will decrease the speed of the motor corresponding to that side so that it straightens out the robot. If the robot veers too quickly to one side so that both sensors detect white, it remembers the side of the robot that last detected the black line and will veer back towards it. This is done using the “lastDetected” flag. The value 400 was determined through testing as an estimated midpoint between white (~50) and black (~800) values.

Lastly, the avoidLine functions are similar to followLine() except the idea is to keep the robot on one side of the line instead of following on top of the line. It works by reading just the outside line sensor and if it detects black at any point, veer inwards to stay within the opponent’s end zone. These functions, along with the entire source code can be seen in Appendix B.

Competition Performance

Given our strategy, we were skeptical on whether or not the robot would be successful against the other contenders, since the basis of the Street Sweeper’s functionality was inherently dependent on the other teams’ strategies.

It became apparent early on that many of the other teams, as we had expected, were very focused on collecting and stacking cans, developing an effective, yet lumbering machine that managed to find cans and stack them well, but could not possibly keep up with the speed of our robot. There was one particular robot that had the same idea as ours, though, which provided for an interesting challenge, since we had not planned for someone to try the same methods as us.

During the initial set-up phases, we were simply testing how the robot performed on the field without competitors, including fine-tuning of the line sensors and regulation of tire speed for turns. We decided an incremental approach would be the most effective way to alter the programming, so we had a process of testing a short line of code, altering it slightly, then re-testing and continuing this procedure until we were satisfied with the results. This proved to be tedious at times, but essentially effective.

We eventually came to a point where we were satisfied with the robot's programming, which was approximately a half an hour before the competition. While other teams frantically scrambled to make final changes, the Street Sweeper lay in waiting.

Our first match, the opponent's robot had a very intricate pressurized-air system in order to pick up cans and stack them. However, they had tried to fit in too many features in the end, and their robot crashed into a wall, then promptly decided to stop moving.

This would seem to be the theme of the day, with several robots either not being able to function as desired, or simply making mistakes. The Street Sweeper was also effective in the sense that it managed to steal several cans from opponents, or knock their cans out of their end zone, at the very least. We had a few issues with battery drainage, but we managed to make it to the Championship match, losing in a narrow 14-9 battle, and taking 2nd place overall (and holding the position as top-performing active student team, as the winners were alumni).

Conclusion

In conclusion, we successfully built a working autonomous robot that was able to fulfill our design requirements. The robot also satisfied all the constraints set forth in the competition rules and performed as we expected. However, there were a couple things we could have done differently in building the robot. First, we could have made the robot much taller to give it more mass, as there were a few instances where the other robots would simply push poor Street Sweeper around the arena. Also, because of our unforeseen power issues, we should have either bought an extra backup battery or used a larger battery in the first place. Going into the championship round, our battery was completely drained after only two rounds, which delayed the start of the final round significantly.

References

[1] Arduino Servo Library, http://arduino.cc/en/Reference/Servo
[2] Romeo All-In-One Controller Manual, http://www.dfrobot.com/wiki/index.php?title=DFRduino_Romeo-All_in_one_Controller_%28SKU:DFR0004%29
[3] QRD1114 Line Sensor Datasheet, http://www.fairchildsemi.com/ds/QR/QRD1114.pdf
[4] Pololu Robotics and Electronics, http://www.pololu.com/
[5] DFRobot Online Robot and Hardware Shop, http://www.dfrobot.com/

Appendix A.

Competition Rules and Course Specification:
Version 1.3 (3/27/12)
This year's competition is a head-to-head double elimination tournament where the object of the competition is to collect small cans and push them into an end zone.

Competition Date: April 14, 2012
Competition Location: Mott Gym, Cal Poly Campus
Registration Deadline: TBD

The competition is organized by the Cal Poly Eta Kappa Nu chapter and the CPE Society. Teams are required to register with their intent to compete in Roborodentia. Registration forms will be made available.
Note: Rules are subject to minor updates and clarifications. Any changes will be announced and noted at the bottom of this page.
Download the competition layout here: Course Sketchup file
1. Course Specifications (see attached diagrams for more details and dimensions)
1.1 The entire course is 8' wide x 8' long with 4" high walls around the edges.
1.2 The black lines shown on the playing field are strips of 3/4" black masking tape.
1.3 Each team has an end zone which is width of the playing field.
1.4 A center dividing wall is located between the two sides of the playing field. The center portion of the wall will be removed 60 seconds into a match.
1.5 The cans are 3 oz. cat food cans (Fancy Feast brand). The cans on each side of the field will be colored. A side will contain either all red cans or all blue cans.
1.6 The cans will be spray painted using Rust-oleum Gloss Sunrise Red and Rust-oleum Gloss Royal Blue. The bottom of the cans will not be painted.
2. Robot Specifications
2.1 Robots must be fully autonomous and self-contained.
2.2 Robots must be 12" x 12" or smaller at beginning of the match, but may autonomously expand after the match begins. At any point during a match, a robot may not be larger than 14" x 14". There is no height limitation.
2.3 A robot may not disassemble into multiple parts.
2.4 Robots may not use any RF wireless receivers/transmitters during the competition.
2.5 Robots may not damage the course or the contest cans.
2.7 Adhesives may be used to pick up cans, but the cans may not be modified in any way. A can must be completely free of residue after it has been picked up.
2.8 If a robot has RF wireless components on-board, the contestant will be required to notify the judges before the competition, and be able to demonstrate that the wireless components are

not used. If RF components are found on-board that were not declared, or declared nonoperational when active, it will be grounds for immediate disqualification.
2.9 Intentionally jamming an opponent's sensors is not allowed.
3. Competition Regulations
3.1 Each side of the field will initially contain 14 cans.
3.2 A robot starts a match in any orientation and location in its end zone.
3.3 A can is scored if the can is resting only on the playing field and the can is entirely within the end zone. A scored can may touch the scoring robot, but may not rest on the robot for support.
3.4 A can is “stacked” and scored if it is resting only on other scored cans. No part of the stacked can may rest on the playing field or a robot in order to be scored.
3.5 The black tape bordering an end zone is considered part of the end zone.
3.6 The center portion of the dividing wall will be removed when a robot scores all of its cans or 60 seconds into a match, whichever comes first.
3.7 If a contest can goes off the course, then the can is out of play with no penalty assessed.
3.8 Robots will be randomly seeded on the morning prior to the event.
3.9 The tournament will be run in a double elimination format.
3.10 A match will last 3 minutes.
3.11 If both teams agree, the match may end prior to three minutes.
3.12 At the end of a match, the robot with more points in the final configuration wins that match.
3.13 One false start is allowed per team per match. A false start must be requested before the team's robot touches a can and within 3 seconds after the match starts.
3.14 A 3 second tone countdown will signal the start of a match. Contestants must start the robot during this period by pressing only 1 button 1 time. Contestants may not touch a robot during a match. Touching a robot ends the run for that robot and the robot keeps all points up to that instant.
3.15 If both robots are entangled for 6 seconds, a referee will tell each team to restart their robot from their respective end zone. The match clock will continue to run during a restart. Both teams must restart their robots (by pressing a button within 10 seconds of the referee’s signal), otherwise the match ends for a robot.
3.16 If a robot travels on to the opponent's side of the playing field and causes the robots to be entangled for 6 seconds, all cans in possession by the offending robot will be required to be released. Those cans may be placed in any location (unstacked and not in an end zone) on the playing field (determined by the team not causing the entanglement). Cans in possession by the robot not causing the entanglement may optionally be removed (otherwise must remain the configuration at ending time of entanglement). The robot not causing entanglement must completely leave and return to the end zone in order for those cans in its possession to be scored.
3.17 In order for a can to be scored, a judge must be able to laterally remove the can without interference from the scoring robot. A can that satisfies this property may not be surrounded by more than 180 degrees at the time of score calculation. All cans above a can that cannot be laterally extracted will also NOT be scored.
4. Scoring
4.1 A team’s can located in a robot's end zone will be worth 1 point. If cans are stacked on other cans, the score for a can will increase by 2 points for each can in a stack. Can scoring will be: 1 point, 3 points, 5 points, and so on based on the height of a can in a stack. For example, a stack of 3 cans is worth 9 total points.
4.2 Opponent cans that reside in a team’s end zone are also scored based on height in a stack: 2 points, 4 points, 6 points, and so on.
5. Penalties
5.1 A robot that attempts to damage an opponent's robot will be disqualified for that match.

5.2 A robot may not have weaponry designed to damage an opponent's robot.
5.3 Robots that do not leave their end zone within the first 20 seconds of a match will be considered inoperable and will forfeit the match.
5.4 If both robots have not moved for 60 seconds (at any time during a match), the match will end.
5.5 If a robot exceeds the size restrictions during a match, the match ends for that robot. The opponent robot may continue the match.
6. Tie breakers
In the event of a tie, the following tie breakers (listed in order below) will be used to determine a winner:
1. Whichever robot scores the tallest stack. If both teams have a stack of the same height, that pair of stacks will be excluded when determining this tiebreaker.
    2. Whichever robot scores more opponent cans.
    3. Whichever robot moves more cans located at the intersections of the black tape.
    4. Whichever robot is not located in their own end zone at the end of the match.
    5. Coin toss
7. Contestant eligibility
Anybody may enter Roborodentia XVII. If the winning team does not have a current Cal Poly student as a member, $200 of the 1st place prize money will be donated to the highest placing team with a student member.
8. Prizes
Below are the prize levels:
1st Place - $1,000
2nd Place - $600
3rd Place - $400

Appendix B.

```
/*Source code for all robot movement
Roborodentia 2012
Street Sweeper
by Jason Miller
*/

#include <Servo.h>

Servo servo1;
Servo servo2;
int pos = 0;
int analogPin = 1;
int analogPin2 = 2;
int lastDetected = 0;
int val;
int left, right;
boolean finished;
//Standard PWM DC control
int E1 = 5;      //M1 (Right wheel) Speed Control
int E2 = 6;      //M2 (Left wheel) Speed Control
int M1 = 4;      //M1 (Right wheel) Direction Control
int M2 = 7;      //M2 (Left wheel) Direction Control

void stop(void)                    //Stop
{
  digitalWrite(E1,LOW);
  digitalWrite(E2,LOW);
}
void advance(char a,char b)             //Move forward
{
  analogWrite (E1,a);  //PWM Speed Control
  digitalWrite(M1,HIGH);
  analogWrite (E2,b);
  digitalWrite(M2,HIGH);
}
void back_off (char a,char b)           //Move backward
{
  analogWrite (E1,a);
  digitalWrite(M1,LOW);
  analogWrite (E2,b);
  digitalWrite(M2,LOW);
}
void turn_R (char a,char b)             //Turn Left
{
  analogWrite (E1,a);
  digitalWrite(M1,LOW);
  analogWrite (E2,b);
  digitalWrite(M2,HIGH);
}
void turn_L (char a,char b)             //Turn Right
{
  analogWrite (E1,a);
  digitalWrite(M1,HIGH);
  analogWrite (E2,b);
  digitalWrite(M2,LOW);
}
void setup(void)
{
  int i;

  for(i=4;i<=7;i++)
          pinMode(i, OUTPUT);
  Serial.begin(9600);  //Set Baud Rate
  Serial.println("Run keyboard control");
  finished = false;
  /*Set up starting positions for servos*/
  servo1.write(80);
  servo2.write(10);
  servo1.attach(13);
  servo2.attach(12);
```

```
}
void loop(void)
{
  int i = 0;

  while (finished == false)
  {
          followPath();

          finished = true;
  }
  stop();
}
void raiseArms()
{
  servo1.write(0);
  servo2.write(90);
  delay(2000);
}
void lowerArms()
{
  int i;
  for (i = 0; i < 80; i++)
  {
          servo1.write(i);
          servo2.write(90 - i);
          delay(15);
  }
  //servo1.write(80);
  //servo2.write(20);
  delay(2000);
}

void followPath()
{
  left = analogRead(analogPin);
  right = analogRead(analogPin2);
  /*Move forward to first black line, then turn right*/
  while (left < 300 && right < 300)
  {
          left = analogRead(analogPin); //Read from left line sensor
          right = analogRead(analogPin2); //Read from right line sensor
          advance(225,225);
  }

  stop();
  delay(500);
  turnRight (825);
  /*Follow black line to the opposite side of the field*/
  followLine();
```

```
stop();
delay(1000);

/*Turn towards endzone, then move forward*/
turnRight(1000);
moveForward(1000);

raiseArms(); //dropping off the cans

turn_180();
lowerArms();

/*Move to the second black line and turn left*/
left = analogRead(analogPin);
right = analogRead(analogPin2);
/*Move forward until line sensors detect the next black line*/
while (left < 300 && right < 300)
{
        left = analogRead(analogPin);
        right = analogRead(analogPin2);
        advance(225,225);
}
stop();
advance(255,255);
delay(300);
left = analogRead(analogPin);
right = analogRead(analogPin2);
/*Move forward until line sensors detect the next black line*/
while (left < 300 && right < 300)
{
        left = analogRead(analogPin);
        right = analogRead(analogPin2);
        advance(225,225);
}
stop();

turnLeft(825);

/*Follow second black line across the field*/
followLine();
stop();
moveBackward(250);

/*Return to endzone to drop off cans*/
turnLeft(825);

moveForward(2800);
raiseArms();

turn_180();
```

```
lowerArms();

/*Advance to second black line*/
left = analogRead(analogPin);
right = analogRead(analogPin2);
/*Move forward until line sensors detect the next black line*/
while (left < 300 && right < 300)
{
        left = analogRead(analogPin); //Read from left line sensor
        right = analogRead(analogPin2); //Read from right line sensor
        advance(225,225);
}
stop();
advance(255,255);
delay(300);
left = analogRead(analogPin);
right = analogRead(analogPin2);
/*Move forward until line sensors detect the next black line*/
while (left < 300 && right < 300)
{
        left = analogRead(analogPin);
        right = analogRead(analogPin2);
        advance(225,225);
}
stop();
turnRight(825);

/*Advance forward while veering downward towards the endzone*/
advance(200, 255); //ARCH
delay(6000);
moveBackward(500);
raiseArms();

turn_180();
lowerArms();

/*Advance to opponents endzone*/
moveForward(8000);
moveBackward(250);
turnRight(825);

/*Move backwards inside opponents endzone to knock any stacks over*/
avoidLineBackward();
/*Now move foward to steal opponent's cans*/
avoidLineForward();
/*exit opponent's endzone*/
turnRight(825);

/*Maneuver around middle wall and return to our endzone*/
moveForward(4000);
moveBackward(250);
```

```
  turnRight(825);
  moveForward(3000);
  turnLeft(825);
  moveForward(4500);
  raiseArms();
  moveBackward(250);
  turn_180();
  lowerArms();

  /*Make one more pass at opponent's endzone*/
  moveForward(8000);
  moveBackward(250);
  turnLeft(825);
  moveForward(4000);
  moveBackward(250);
  turnLeft(825);
  /*This time, follow left wall back home*/
  moveForward(4000);
  moveBackward(250);
  turnLeft(825);
  moveForward(3000);
  turnRight(825);
  moveForward(5000);
  moveBackward(250);
  raiseArms();
  turnRight(6400);

  lowerArms();
  raiseArms();
  lowerArms();

}

/*turn right at full speed for <time> milliseconds*/
void turnRight(int time)
{
  turn_R(255, 255);
  delay(time);
  stop();
}

/*turn left at full speed for <time> milliseconds*/
void turnLeft(int time)
{
  turn_L(255, 255);
  delay(time);
  stop();
}

/*advance at full speed for <time> milliseconds*/
void moveForward(int time)
```

```
{
  advance(255, 255);
  delay(time);
  stop();
}

/*move backwards at full speed for <time> milliseconds*/
void moveBackward(int time)
{
  back_off(255, 255);
  delay(time);
  stop();
}

void turn_180()
{
  /*Turn 180 degrees*/
  turn_R(255,255);
  delay(1500);
  stop();
}
/*Function that follows a black line from one side to the other*/
void followLine()
{
  int i;
  for (i=0; i<7000; i++)
  {
        left = analogRead(analogPin);
        right = analogRead(analogPin2);

        if (left < 400 && right > 400)
        {
        //Serial.println("Veering left");
        advance(220, 255); //Decrease right motor speed
        lastDetected = 1;
        }
        else if (right < 400 && left > 400)
        {
        //Serial.println("Veering right");
        advance(255, 220); //Decrease left motor speed
        lastDetected = 2;
        }
        if ((left < 400 && right < 400) && lastDetected == 1)
        {
        //Serial.println("Veering left, lastDetected");
        advance(220,255);
        }
        else if ((left < 400 && right < 400) && lastDetected == 2)
        {
        //Serial.println("Veering right, lastDetected");
        advance(255,220);
```

```
          }
          else
          {
          advance(255,255);
          }
          delay(1);
  }
}

/*Function to keep robot within oppenent's endzone moving backwards*/
void avoidLineBackward()
{
  int i = 0;

  for (i = 0; i < 4000; i++)
  {
          right = analogRead(analogPin2);
          if (right > 300)
          {
          back_off(255,235);
          }
          else
          {
          back_off(255,255);
          }
          delay(1);
  }
  stop();
  //advance(255, 250);
  //delay(7000);
}

/*Function to keep robot within opponent's endzone moving fowards*/
void avoidLineForward()
{
  int i = 0;

  for (i = 0; i < 7000; i++)
  {
          right = analogRead(analogPin2);
          if (right > 300)
          {
          advance(255,235);
          }
          else
          {
          advance(255,255);
          }
          delay(1);
  }
  stop();
  //advance(255, 247);
  //delay(7000);
}
```

Appendix C.

Gantt Chart

	1/24	1/31	2/7	2/14	2/21	2/28	3/6	3/13	3/20	3/27
Finalize idea, purchase needed parts.	🟦	🟦								
Build prototype			🟦	🟦	🟦					
Programming					🟦	🟦	🟦			
Testing/Debugging				🟦	🟦	🟦	🟦	🟦	🟦	🟦