

Robot X

Computer Engineering

California Polytechnic State University

San Luis Obispo, CA

Roborodentia 2012

Alan Truong, Alex Haag

Introduction

The objective of this project was to design and build an autonomous robot to compete in the 2012 Cal Poly Roborodentia competition. While the robot was required to meet the specifications of the competition, those requirements were very open, and guided, not hindered, the design of the robot. The goal of the competition was to design a robot that would go onto a playing field and autonomously navigate and collect as many cans of cat food from the course, and return them to a goal zone, while another robot does the same thing on the other half of the field. Points are awarded based on number of cans, which team the can belonged to, and if the cans were stacked. Our design focused on collecting cans from our own side quickly and efficiently, stacking them inside of the robot, and then returning a complete stack to the goal, for a large point total.

Design

Team Process Overview

The construction of Robot X was based on a modular design. Each component were build independent of each other and designed to be fairly easily taken apart. This includes the ability to fully disassemble the conveyor belt system from the robot with the use of two screws, as well as screws for securely other components. The first component to be constructed was the robot chassis which was a 12 x 12 particleboard. This was followed by construction of the conveyor belt system, swinging arms, and trap doors. During each construction state, heavy testing was conducted to ensure reliability and repeatability of Robot X. The last component that was developed for Robot X was electronics and programming.

System Architecture

Vision

The robot relies on multiple IR range sensors placed throughout the robot to determine its position within the playing field. These sensors are located on the front, sides, and rear of the robot as seen in Figure 1. These sensors are capable of measuring a continuous distance range of 20cm to 150cm, while consuming 33mA at a supply voltage ranging from 4.5 to 5.5V and outputting its reading via analog output [1]. A total of three different IR sensors were used for Robot X which consist of 5cm short range digital IR range sensors, mid range IR sensors and long range IR sensors. A more detailed schematic of the sensors may be viewed in appendix B.

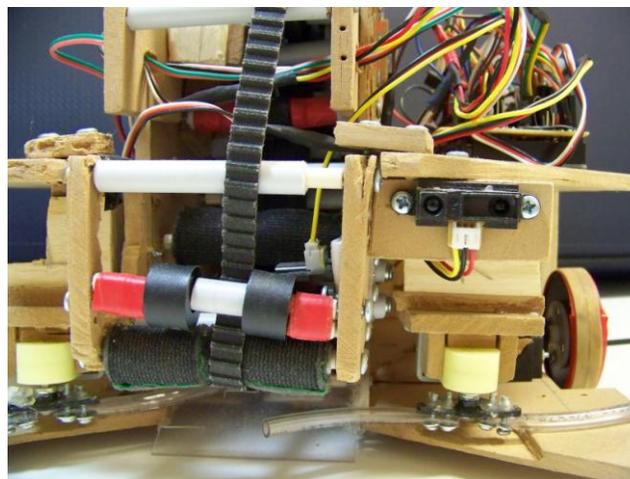


Figure 1 - IR Sensor Placement

In addition to the IR sensors, the robot utilized a optical mouse sensor located off center to the side of the robot as seen in Figure 2. The optical sensor used was an V7 3 Button PS2 Optical Mouse . An PS2 mouse was chosen since it provided a simple interface with the Arduino Mega's existing library with minimal to no additional hardware requirements [2]. An modern USB mouse would require additional hardware in order to interface with the USB protocol which was undesirable. The PS2 protocol worked with the Arduino through simple modifications of SPI. With this communication the optical mouse sensor relayed information about its change in position in both the X and Y coordinate relative to the last time the mouse was read. With the sensor off center, the optical mouse was used to measure rotational change of the robot, thus ensuring the robot turned exactly as needed.

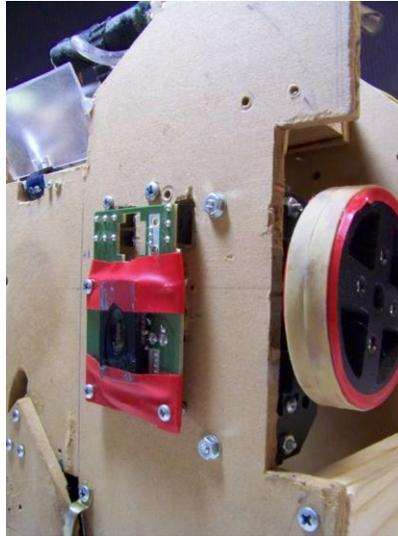


Figure 2 - PS2 Mouse Sensor

Drive System

Robot X used two continuous rotational servos to provide its driving capabilities. Servos were chosen as the main driving system since it provided high torque with a simple interface to the Arduino [3]. In addition to the two driving wheels, Robot X had four metal ball bearing casters located in each corner of the robot to provide additional support and balance. In Figure 3 below, the two wheels and front casters may be seen.

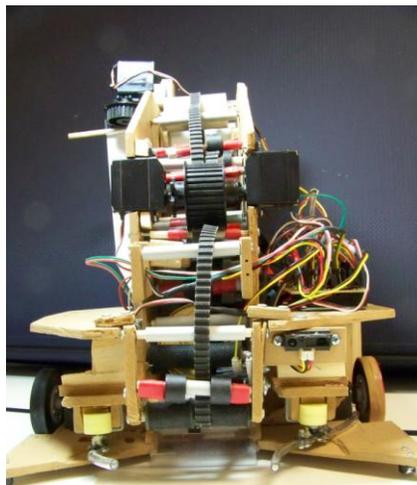


Figure 3 - Robot X Front View

Conveyor Belt and Swinging Arms

An conveyor belt system would be useless if cans never got to the entrance, so there needed to be a way to push the cans up to the conveyor. This was accomplished using two motors located in the front of Robot X. Attached to these motors are flexible vinyl tubing which swing inward to push the cans towards the entrance of the conveyor system as seen in Figure 4.

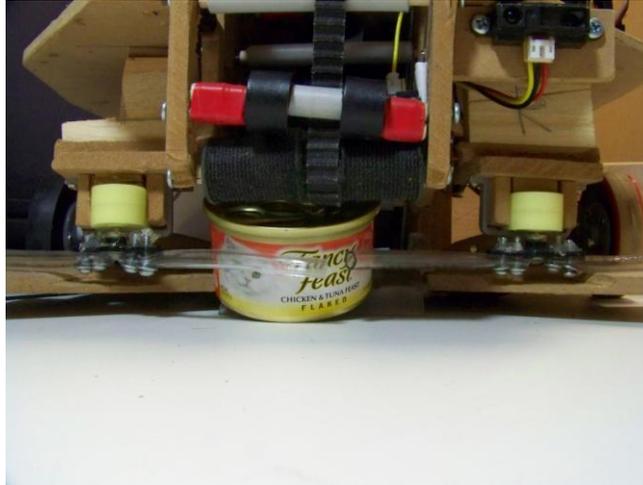


Figure 4 - Swinging Arms

With the swing arms in place, the robot used a two sided conveyor belt system to carry the cat food cans up to be stacked in the can compartment. This system provided a contact points in both the top and bottom of the can to be pulled up. To provide enough torque to carry multiple cans a Digital Servo was used on the bottom conveyor belt that provided over 250 oz-in torque, over 6 times the torque of a traditional servo [4]. In addition, two normal continuous rotational servos were placed to provide enough torque for the top conveyor belt as seen in Figure 5 and Figure 6.

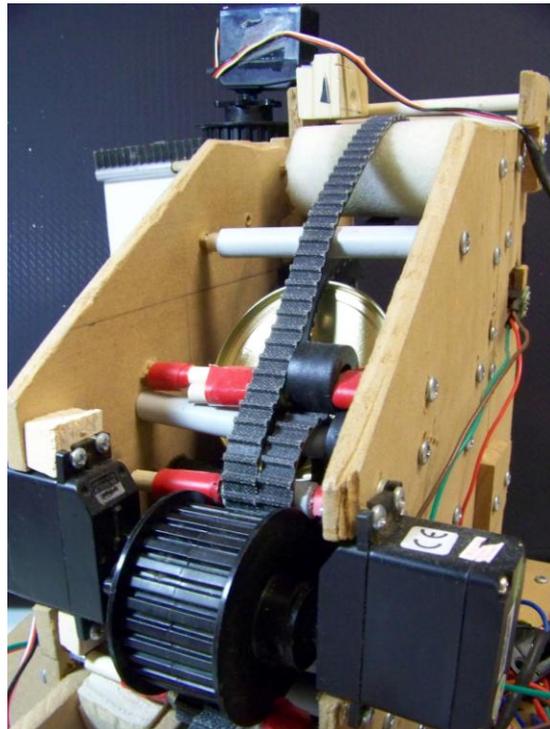


Figure 5 - Conveyor Belt System Outside View

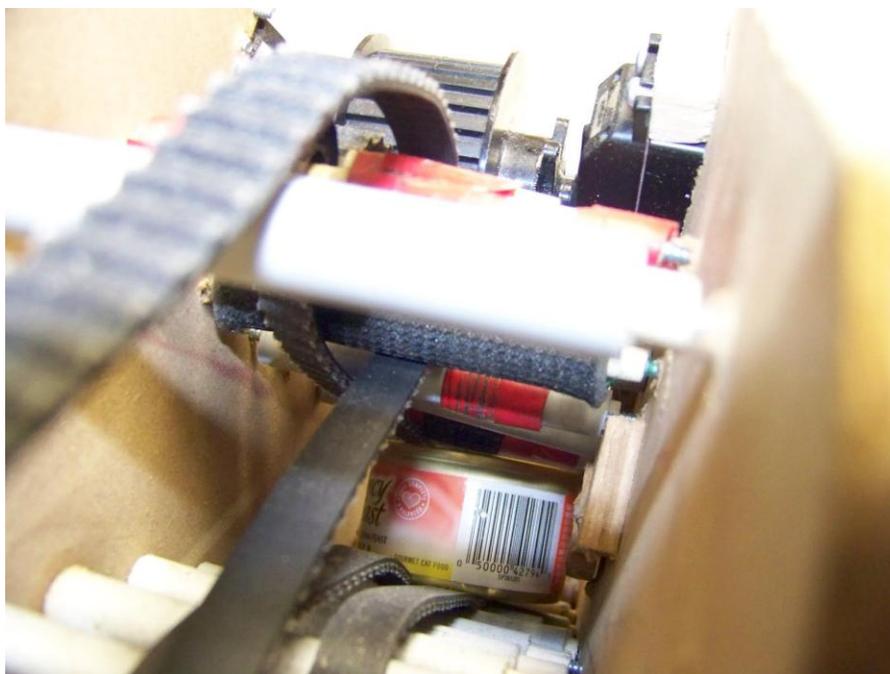


Figure 6 - Conveyor Belt System Inside View

Can Compartment and Doors

Once the can compartment became full, Robot X needed a way to release the cans with minimal interference to the cans to prevent accidental knock down of the stacked cans. This was accomplished using a combination of a trap door located at the bottom of the stack of cans, and a slide door that slid open. The trap door as seen in Figure 7 uses a standard rotational servo facing downward towards the ground. The trap door was attached through three contact points on the servo as well as an additional supports to handle the stack load through a screw and groove in the robot.



Figure 7 - Trap Door

The second component of the can compartment was the slide door that relied on a rail system located at the top of Robot X as seen in Figure 8. The slide door used a continuous servo to open and close.



Figure 8 - Slide Door

To drop off the cans, Robot X would first open the trap door so that the cans were now resting on the playing field. Next, the slide door was opened and Robot X moved forward, leaving the stacked cans behind it as seen in Figure 9.



Figure 9 - Drop off

Controller and Power

The Arduino Mega was chosen as the controller for Robot X. During the testing stages of Robot X an Arduino Uno was used, but was upgraded to an Mega due to the growing PWM channel requirements with the multiple servos within the robot [5]. The robot was powered off of five rechargeable AA batteries to provide 6.0V. Originality Robot X used four rechargeable AA batteries (4.8V), but due to increasing need for more torque for the conveyor belt system, the voltage was increased.

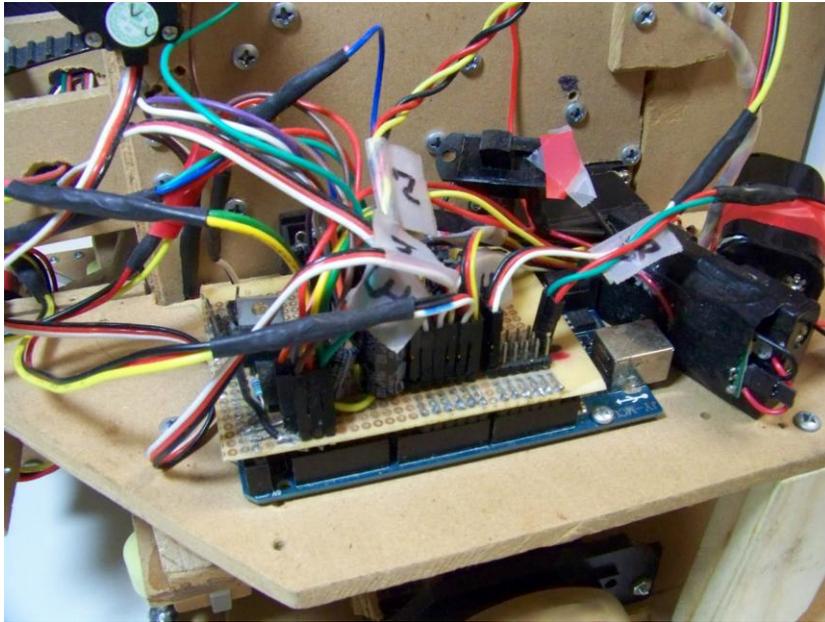


Figure 10 - Controller

Hardware Block Diagram

An overall hardware block diagram may be viewed in Figure below. On the left side of the diagram are all components related to Robot X's vision. This includes IR sensors for measuring distances from the front, side and rear of the robot as well as sensors to detect full can loads of the robot. In addition, an optical mouse sensor was used to measure rotational movements of the robot to ensure exact turns. All sensors were connected to an Arduino Mega through digital I/O pins or analog I/O pins and were powered off of the on board 5V regulator of the Arduino Mega which was powered from the 5 rechargeable AA batteries. On the right side of the diagram are all the motorized components of the robot. This includes the driving servos, conveyor belt servos, swinging arm motors, and can compartment doors. All motorized components were controlled by the Arduino Mega's PWM I/O pins and powered from five rechargeable AA batteries.

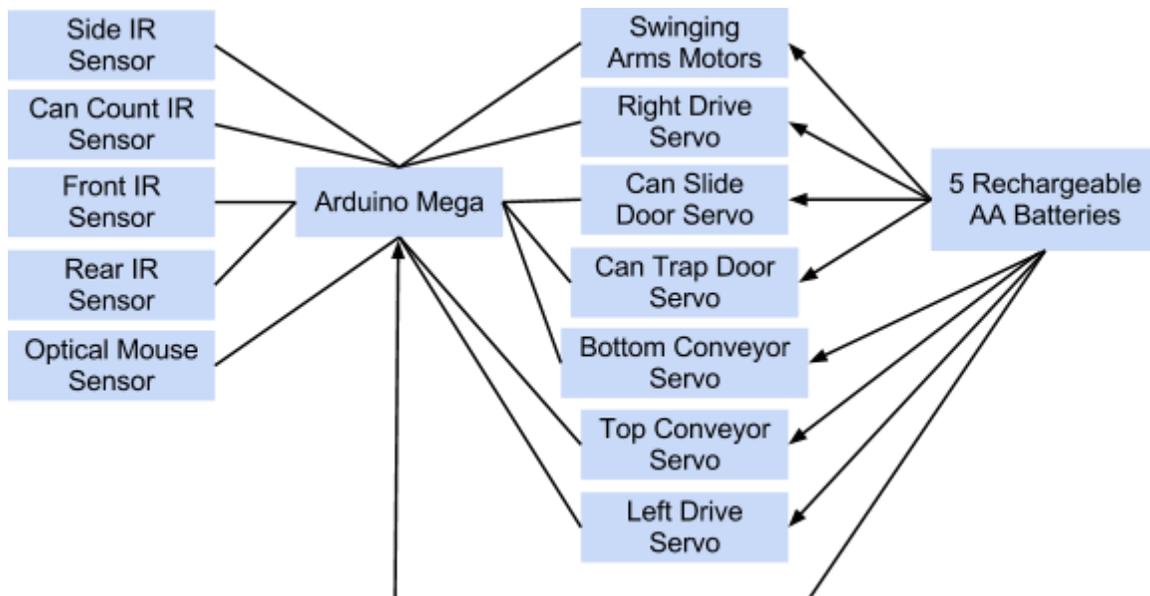


Figure 11- Hardware Block Diagram

Software Algorithms

The robot software was created using a state machine to determine its course of action based on its position within the playing field. This state machine was designed with the assumption that the robot would be able to collect all cans before the wall was removed. A general software flowchart of Robot X may be viewed in Figure 12 with a full source code listing in appendix C.

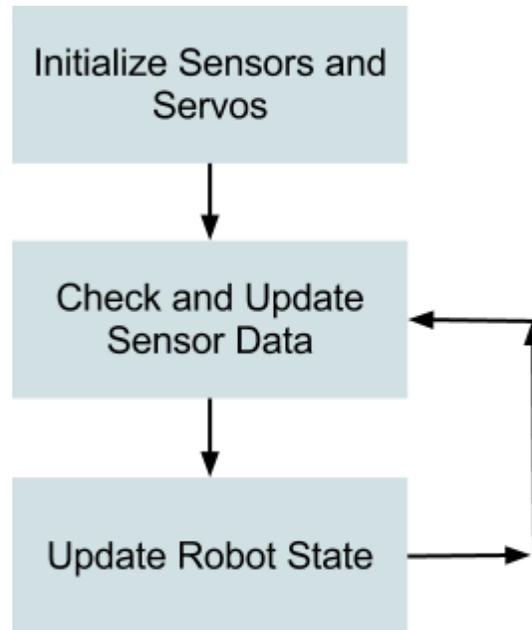


Figure 12 - Robot X General Software Flowchart

Median Filter

In order to ensure the reliability and accuracy of the data from the various sensors, filtering was used to ignore any outlier data caused by the environment or noise on the power line. This algorithm uses an array of the of between 7 to 51 of the latest sensor readings depending on the sensor. This algorithm may be seen in Figure 13 which uses the 5 latest readings. From these five sonar readings, the filter would sort the array and output the median value which would be 72 in the example from Figure 13. Unlike averaging, median filter output value is not affected by any outlier since those values would be sorted to the edge of the array.

5 Latest Sonar Readings:

0	1	2	3	4
70	73	201	72	71

Sorted Sonar Readings:

0	1	2	3	4
70	71	72	73	201

Median is 72

0	1	2	3	4
70	71	72	73	201

Figure 13 - Median Filter Example

Driving Drift Correction Algorithm

One issue that needed to be address was drift by the robot. That is, the robots inability to go in a straight path over time. This drift was caused by various sources such as differences in servos, wheel shape, wheel slippage, robot weight distribution, and robot balance. To counteract this drift, IR range sensors were placed on the robots side as seen in Figure 14. These sensors were used to measure its distance from the wall. This data was then filtered using median filtering, and checked against a predefined distance value. If Robot X began to drift too far away from the wall, it would correct itself by turning in the wall and vice versa if it drift too close to the wall.

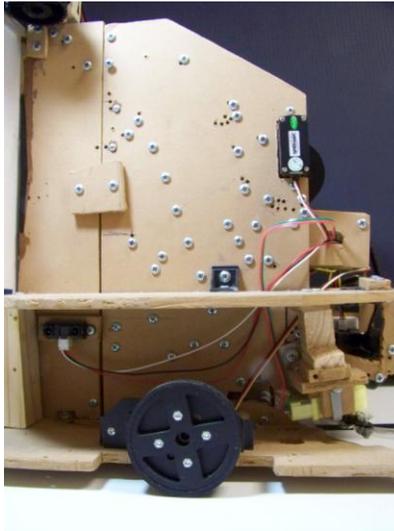


Figure 14 - Side IR range Sensors

Mechanical design

The goal of Robot X was to not only stack cans, but to also stack them as quickly as possible, thus preventing the opponents from stealing the cans. With this particular design criteria in mind, Robot X was designed using a conveyor belt system rather than a claw, thus enabling it to pick up cans while continuing its movement throughout the playing field. Initial design sketches of Robot X may be seen in Figure 15.

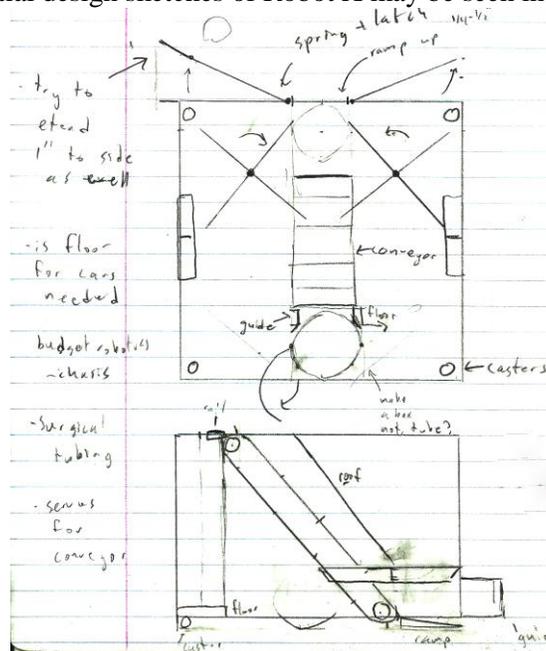


Figure 15 - Robot X Design Sketches

Chassis

The chassis of the robot was built out of ¼" particleboard. This was chosen for its low price, workability, and strength. Due to limited access to a machine shop, we opted for particle board, since it could be cut with a hand saw, and easily drilled for mounting screws. The particle board did prove useful for these purposes, but had drawbacks. It was very heavy, most likely accounting for half of the robots total weight, and the layers of compressed sawdust were prone to peeling apart, and meant that the board could only be drilled into the surface, not into the edges. We were also able to quickly adjust parts, trim edges, and cut out holes in the bottom of the chassis by using hand tools, without needing to completely disassemble the robot.

Conveyor Belt

The main components of the conveyor belt were the rollers, which were made from wooden dowels covered by PVC pipe, attached to a structural wall by screws, and the belts, which were rubber timing belts. The roller assembly presented many problems, as the dowels tended to split when the holes for screws were drilled, and were not of uniform diameter, so needed to be smoothed, to allow the PVC pipe to freely rotate over it. This system would have been re-hauled, given more time, to use something easier to work with. The screws also proved problematic, as the conveyor belt assembly used over 150 screws itself, contributing to the robots weight, and making assembly difficult.

The belts, however, worked well. The timing teeth on them allowed the top belt to grab cans effectively, and allowed the bottom belt to be connected to a timing pulley and not slip. The only problem was that they were not an adjustable diameter, and two had to be used for each side of the conveyor.

Swing Arms

The swing arms were made with flexible plastic tubing, which allowed them to bend. The material was chosen because it was stiff enough to pull and push cans, but would bend out of the way if the can stop moving, which happened often as the cans reached the entrance to the conveyor. It also allowed the motors to be placed lower to the ground, which made them better at guiding cans, without having to worry about them hitting the sides of the robot, as they bent out of the way when they hit it.

Can Compartment Door

The can compartment main door was made from a piece of polystyrene sheet, because of its lightweight and flexible nature. The door needed to slide back and forth, and flex outwards as more cans were deposited, which the 1mm thick plastic made possible. At the point in the design that we implemented it, weight had become a concern, so the lightweight plastic was also a plus.

Design decisions

Our design focused on two main design goals. Those were to collect cans as quickly and efficiently as possible, and to automatically stack them and store them inside the robot until a full stack was made, then to drop them at the goal.

The first design goal, to efficiently and quickly collect cans, was accomplished by the swinging arms and the path finding we used. The swinging arms were designed to scoop in any can in the path of the robot, a 12" area moving forward, and direct them towards the center of the robot. Once there, the swinging arms would then grab each can and force it into the conveyor. The arms were also flexible, so that it could accommodate more than one can in the collection area at the front of the robot at a time. This system lead to the robot being able to insert cans into the conveyor at about one can every three seconds, and was able to insert every can in a single row (about 7 cans) without jamming, as it moved along the row.

The other aspect of the speed in which the robot was able to collect cans came from its path finding software and vision. We opted not to use line following, and instead used a combination of IR sensors and an optical mouse for vision. Using the information from these sensors, we were able to program the robot to take a spiral path across the course. This system allowed the robot to move quickly along the course in a preset pattern, and not spend time thinking about where to go based on the course lines, or cans on the field.

The other design goal for the robot was to stack the cans automatically and return completed stacks to the goal. This was accomplished with the conveyor belt and can holding area. The conveyor belts constant movement, combined with the cans being fed in from the swing arms, meant the robot would grab onto a can from any position it came into contact with the robot's front, and be moved up the robot to the top of the stack holding area. By holding the cans inside the robot in a stack, we were able to perform two tasks at once, as the robot stacked and collected simultaneously. This contributed to the robot's speed, and allowed us to create a stack without using complicated servo arms to stack cans mechanically. Once the robot had a full stack, it would return to the goal and deposit the cans. This was the only time the robot was not spending actively collecting cans, making the time it spent on the course very efficient.

Conclusion

While our robot was able to perform well in testing, during the competition a few problems led to it not performing as expected. Before the competition, while testing the robot, it was reliably returning one stack of 7-8 cans, and then returning to the field and retrieving more, although it was not as accurate on the second set. We had worked out most of the problems with turning, reversing, and dropping off the cans.

However, the day before the event we began testing the robot thoroughly, and this is most likely what caused the problems the day of the event. Between the constant handling and adjusting of the robot, its delicate wiring, and the wear on its components from the testing, things started breaking.

For instance, less than an hour before our first round, the swing door that drops the cans from the holding area to the ground was not able to move while fully loaded with cans, even though it had worked during the rest of our tests the day before. Additionally, the large current draw of the motors meant that performance was varied based on remaining battery power, and they needed to be changed frequently. The largest factor in the robot not performing during the competition, however, was a bad solder joint for the rear IR sensor, which caused the robot to lose track of its position during the return and drop off phase. With a little more time, and better testing procedures, these errors could have been avoided, but with the time constraints placed on the competition, we were not able to rectify these problems in time.

What We Would Do Differently

After running into multiple issues with Robot X during its construction, there were several design decisions we would have done differently as well as how we managed our time. The biggest issue we encountered was how we managed our time for the construction of Robot X with too much time focused on the mechanical aspect of Robot X and only a week or two left for the electrical components and system testing. Had we partitioned more time for electrical development, most of the bugs related to Robot X such as a bad sensor solder joint could have been caught early and a solution could have been found. One such issue was Robot X performance sensitivity relative to battery charge levels. This began to become an issue since Robot X was unable to smoothly pull up cans in its conveyor system on low battery levels with an unregulated battery source. This issue could have been minimized using a UBEC regulator with a high capacity battery such as the 7.4V Lithium 8800mAh battery available in the capstone lab rather than 5 rechargeable AA batteries. Another issue that was

encountered was the trap door failure during the competition. This was caused by overloading the trap door with 8 cans. This issue was solved by adding a thin plastic sheet on top of the trap door to reduce friction between the cans and the door allowing it to swing open with less torque. This solution was not discovered until after the competition. Given more time to test under full loads, this issue could have been solved before the competition.

Budget and justification

Robot X did not have a specified budget, but the goal was to utilize all available resources before making any purchases. The majority of the cost for Robot X was a result of electronic components such as sensors and servos. Nearly all components bought were used, with the exception of the reflectance sensor array used for line following.

Bill of materials

Supplier	Item	Quantity	Unit Price	Total
Pololu.com	180:1 Mini Plastic Gearmotor 90-Degree 3mm D-Shaft Output	2	5.49	10.98
	QTR-8RC Reflectance Sensor Array	1	14.95	14.95
	Pololu Ball Caster with 3/8" Metal Ball	4	2.99	11.96
	SpringRC SM-S4303R Continuous Rotation Servo	2	12.95	25.9
	Pololu Carrier with Sharp GP2Y0D805Z0F Digital Distance Sensor 5cm	2	6.95	13.9
	S & H	1	3.95	3.95
	Tax	1	0	0
Amazon.com	Timing Belt Fiberglass Reinforced Neoprene, One-Sided, 3/8" Wide x 21" Long x 1/5" Pitch	2	2.89	5.78
	Timing Belt Pulley Nylon Glass Filled, Single Flange, 1/5" Pitch, 3/8" Bore, 1.719" Pitch Diameter	2	3.75	7.5
				0
Amazon.com	Timing Belt Fiberglass Reinforced Neoprene, One-Sided, 3/8" Wide x 20" Long x 1/5" Pitch	2	2.09	4.18
	Timing Belt Pulley Nylon Glass Filled, Single Flange, 1/5" Pitch, 3/8" Bore, 1.592" Pitch Diameter	2	5.29	10.58
				0
Amazon.com	V7 3 Button PS2 Optical Mouse	1	6.84	6.84
				0
dealextreme.com	JY-MCU Arduino MEGA Interactive Media MEGA1280 Development Board	1	23.71	23.71
				0
sparkfun.com	Infrared Proximity Sensor Long Range - Sharp GP2Y0A02YK0F	1	14.95	14.95
	Infrared Sensor Jumper Wire - 3 Pin JST	1	1.5	1.5
	S & H	1	3.95	3.95
				0
amazon.com	Timing Belt Pulley Nylon Glass Filled, Single Flange, 1/5" Pitch, 1/4" Bore, 1.910" Pitch Diameter	2	2.21	4.42
				0
Home Depot	6x1/2 Screws	1	3.72	3.72
	Non-Glare Styrene 8x10.50 plexiglass	1	1.97	1.97
	Tax	1	0.44	0.44

Home Depot	6x3/4 Screws	1	4.24	4.24
	1/4x48 Wooden Dowel	2	0.75	1.5
	Tax	1	0.44	0.44
				0
Home Depot	1/4x5ft Pex Pipe	2	1.78	3.56
	1/4x48 Wooden Dowel	1	0.75	0.75
	Tax	1	0.33	0.33
				0
Home Depot	1/4x5ft Pex Pipe	1	1.78	1.78
	1/4x48 Wooden Dowel	2	0.75	1.5
	Tax	1	0.25	0.25
				0
Radioshack	8AA Battery Holder	1	2.59	2.59
	Tax	1	0.2	0.2
				0
Miners Hardware	Tube Vinyl 16IDx5/16OD	1	0.29	0.29
	Tube Vinyl .17ID x 1/4OD	1	0.19	0.19
	Tube Vinyl .17ID x 1/4OD	1	0.19	0.19
	Tube Vinyl 1/4ID x 3/8 OD	1	0.29	0.29
	Tax	1	0.07	0.07
				0
Radioshack	PC Proto Board	1	3.99	3.99
	TIP31 Transistor	1	1.69	1.69
	1N4004 Diodes	1	1.19	1.19
	Tax	1	0.53	0.53
Miners Hardware	Tube Vinyl .17ID x 1/4OD	5	0.19	0.95
	Tax	1	0.07	0.07
Total				197.77

References

[1] Sharp Corporation. "GP2Y0A02YK0F". Internet: http://www.sparkfun.com/datasheets/Sensors/Infrared/gp2y0a02yk_e.pdf, Dec. 01, 2006 [May 31, 2012].

[2] Arduino Playground. "PS2 mouse interface for Arduino". Internet: <http://www.arduino.cc/playground/ComponentLib/Ps2mouse>, Jul. 18, 2009 [May 30, 2012].

[3] Pololu Robotics & Electronics. "SpringRC SM-S4303R Continuous Rotation Servo". Internet: <http://www.pololu.com/catalog/product/1248>, 2012 [May 28, 2012].

[4] Futaba. "Digital Servos". Internet: <http://www.futaba-rc.com/servos/digital.html>, 2012 [Apr. 23, 2012].

[5] Arduino. "Arduino Mega". Internet: <http://arduino.cc/en/Main/ArduinoBoardMega>, 2012 [Mar. 29, 2012].

Appendices

a. Project Contribution

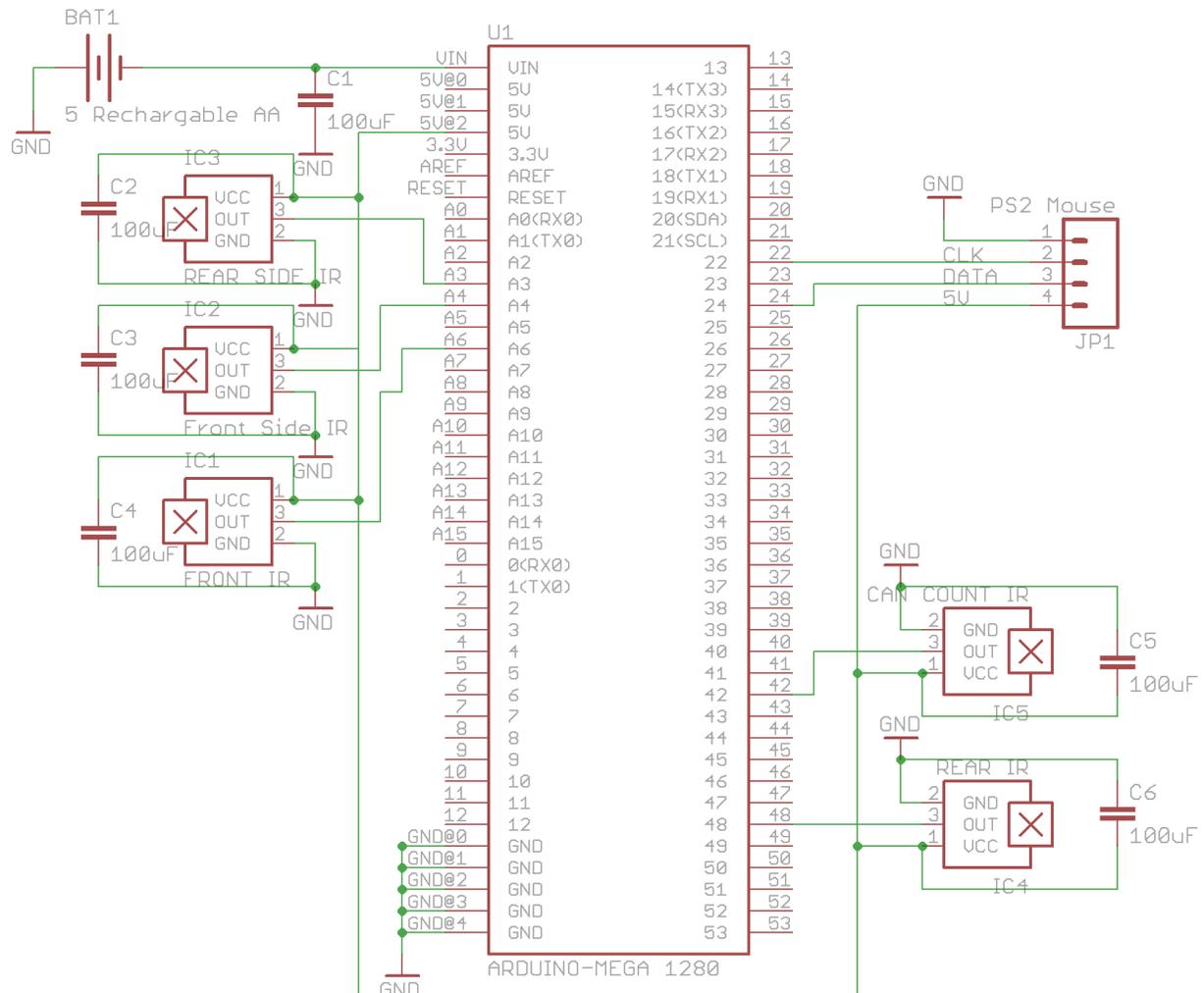
Alan Truong's Contribution:

As team leader, I was responsible for key decisions such as overall design, component chooses, scheduling, and software algorithm. In addition to these roles, I primarily focused on the development and testing of the conveyor belt system of the robot.

Alex Haag's Contribution:

As team support, I was responsible for the original concept designs, and chassis construction and testing. I also helped in refining and testing components, as well as full system testing.

b. Sensors Schematic



c. Robot X Source Code

```

#include <Servo.h>
#include <ps2.h>

// PWM Pins
#define B_CONVEYOR      2
#define T_L_CONVEYOR   3 // relative to if we were staring at its front
#define T_R_CONVEYOR   4 // relative to if we were staring at its front
#define L_DRIVE        5 // relative to if we were driving
#define R_DRIVE        6 // relative to if we were driving
#define TRAP_DOOR      7 // Sliding floor to release can from robot
#define SLIDE_DOOR     8

//Digital I/O
#define SWEEPERS        26
#define CAN_COUNT_SENSOR 42
#define REAR_RANGE_SENSOR 48
#define MOUSE_CLK      22
#define MOUSE_DATA     24

//Analog I/O
#define FRONT_RANGE_PIN 6
#define SIDE_RANGE_PIN  5
#define SIDE_REAR_PIN   3

//Constants
#define NO_CORRECT      0
#define CORRECT        1
#define FORWARD        0
#define STOP           1
#define REVERSE        2
#define ARRAY_SIZE     7
#define SIDE_ARRAY_SIZE 11
#define CAN_ARRAY_SIZE 51
//#define MAX_CAN_COUNT 16
//#define LEFT90        2700
//#define SECOND_LEFT90 2700
#define LEFT90         2600
#define SECOND_LEFT90 2400
#define LEFT180        4800 //changed here
#define RIGHT90        7400
#define FRONT_DIST     115
#define CLOSE_SIDE_RANGE_MIN 500
#define CLOSE_SIDE_RANGE_MAX 520
#define FIRST_SIDE_RANGE_MIN 220
#define FIRST_SIDE_RANGE_MAX 235
#define SECOND_SIDE_RANGE_MIN 190
#define SECOND_SIDE_RANGE_MAX 210
#define SIDE_REAR_MIN   400
#define SIDE_REAR_MAX   420
#define SIDE_REAR_MIN2  420
#define SIDE_REAR_MAX2  440

//STATE MACHINE CONSTANTS
#define START           1
#define FIRST_RIGHT     2
#define FIRST_STRAIGHT 3
#define FIRST_LEFT     4

```

```

#define SECOND_STRAIGHT 5
#define SECOND_LEFT 6
#define THIRD_STRAIGHT 7
#define THIRD_LEFT 8
#define FOURTH_STRAIGHT 9
#define FOURTH_LEFT 10
#define DROP_OFF 11
#define HOME 12
#define REVERSE_TURN 14
#define REVERSE_THIRD_STRAIGHT 16
#define PROTECT 17
#define LEFT_SLOW_REV 1467
#define RIGHT_SLOW_REV 1499
// FORWARD, STOP, and REVERSE servo values for all servos
static int v[9][3] = { {0,0,0}, // Nothing
                      {0,0,0}, // Nothing
                      {85,95,95}, // B_CONVEYOR
                      {100,170,170}, // T_L_CONVEYOR
                      {100,34,34}, // T_R_CONVEYOR
                      {110,90,80}, // L_DRIVE
                      {70,92,112}, // R_DRIVE
                      {115,0,0}, // TRAP_DOOR
                      {70,93,100}}; //Slide_Door

// Global variables
Servo MyServo[9];
PS2 mouse(MOUSE_CLK,MOUSE_DATA);

int frontRangeArray[ARRAY_SIZE];
int rearRangeArray[ARRAY_SIZE];
int sideRangeArray[SIDE_ARRAY_SIZE];
int sideRearRangeArray[SIDE_ARRAY_SIZE];
int canCountArray[CAN_ARRAY_SIZE];
int previousState = HIGH;

int state = START;
int sideRearMin = SIDE_REAR_MIN;
int sideRearMax = SIDE_REAR_MAX;
int index = 0;
int index2 = 0;
int index3 = 0;
int index4 = 0;
int index5 = 0;
int canCount = 0;
long curY = 0;
long destination = 0;
int conveyorState = FORWARD;
int sideRangeMin = CLOSE_SIDE_RANGE_MIN;
int sideRangeMax = CLOSE_SIDE_RANGE_MAX;

void setup(){
  //setup all I/Os
  pinMode(SWEEPERS,OUTPUT);
  digitalWrite(SWEEPERS,LOW);
  pinMode(CAN_COUNT_SENSOR,INPUT);
  pinMode(REAR_RANGE_SENSOR,INPUT);

  //Attach all servos

```

```

MyServo[B_CONVEYOR].attach(B_CONVEYOR);
MyServo[B_CONVEYOR].write(v[B_CONVEYOR][STOP]);
MyServo[T_L_CONVEYOR].attach(T_L_CONVEYOR);
MyServo[T_L_CONVEYOR].write(v[T_L_CONVEYOR][STOP]);
MyServo[T_R_CONVEYOR].attach(T_R_CONVEYOR);
MyServo[T_R_CONVEYOR].write(v[T_R_CONVEYOR][STOP]);
MyServo[L_DRIVE].attach(L_DRIVE);
MyServo[R_DRIVE].attach(R_DRIVE);
MyServo[L_DRIVE].write(v[L_DRIVE][STOP]);
MyServo[R_DRIVE].write(v[R_DRIVE][STOP]);
MyServo[TRAP_DOOR].attach(TRAP_DOOR);
MyServo[SLIDE_DOOR].attach(SLIDE_DOOR);

setArray(sideRangeArray,
SIDE_ARRAY_SIZE, (CLOSE_SIDE_RANGE_MIN+CLOSE_SIDE_RANGE_MAX)/2);
setArray(sideRearRangeArray, SIDE_ARRAY_SIZE, (SIDE_REAR_MIN+SIDE_REAR_MAX)/2);
setArray(canCountArray, CAN_ARRAY_SIZE, HIGH);

for(int i = 0; i < ARRAY_SIZE; i++){
    frontRangeArray[i] = 0;
    rearRangeArray[i] = HIGH;
    canCountArray[i] = HIGH;
}
Serial.begin(9600);
closeCanCompartment();
//startConveyor();
mouse_init();
}

void loop(){
    switch(state){
        case (START):
            if(curY < 2500){ //TODO find exact value
                //Serial.print(curY);
                //Serial.println();
                goForward(NO_CORRECT);
            }else{
                curY = 0;
                destination = RIGHT90;
                state = FIRST_RIGHT;
            }
            break;

        case (FIRST_RIGHT):
            if(curY < destination){
                turnRight();

                if(conveyorState == FORWARD){
                    if(updateCanCount() == LOW){
                        stopConveyor();
                        conveyorState = STOP;
                    }
                }
            }else{
                stopMoving();
                curY = 0;
                sideRangeMin = FIRST_SIDE_RANGE_MIN;
                sideRangeMax = FIRST_SIDE_RANGE_MAX;
            }
    }
}

```

```

setArray(sideRangeArray, SIDE_ARRAY_SIZE, (FIRST_SIDE_RANGE_MIN+FIRST_SIDE_RANGE_MAX
)/2);
    state = FIRST_STRAIGHT;
}
break;

case (FIRST_STRAIGHT):
    goForward(CORRECT);
    if(conveyorState == FORWARD){
        if(updateCanCount() == LOW){
            stopConveyor();
            conveyorState = STOP;
        }
    }
    if(checkFrontRange() > FRONT_DIST){
        stopMoving();
        setArray(frontRangeArray, ARRAY_SIZE, 0);
        curY = 0;
        destination = LEFT90;
        state = FIRST_LEFT;
    }
    break;

case (FIRST_LEFT):
    if(curY < destination){
        turnLeft();

        if(conveyorState == FORWARD){
            if(updateCanCount() == LOW){
                stopConveyor();
                conveyorState = STOP;
            }
        }
    }else{
        stopMoving();
        curY = 0;
        sideRangeMin = CLOSE_SIDE_RANGE_MIN;
        sideRangeMax = CLOSE_SIDE_RANGE_MAX;
        setArray(frontRangeArray, ARRAY_SIZE, 0);

setArray(sideRangeArray, SIDE_ARRAY_SIZE, (CLOSE_SIDE_RANGE_MIN+CLOSE_SIDE_RANGE_MAX
)/2);
        state = SECOND_STRAIGHT;
    }
    break;

case (SECOND_STRAIGHT):
    goForward(NO_CORRECT);

    if(conveyorState == STOP){
        curY = 0;
        state = HOME;
        stopMoving();
        //sideRangeMin = CLOSE_SIDE_RANGE_MIN;
        //sideRangeMax = CLOSE_SIDE_RANGE_MAX;

```

```

setArray(sideRearRangeArray, SIDE_ARRAY_SIZE, (SIDE_REAR_MIN+SIDE_REAR_MAX)/2);
    }else if(updateCanCount() == LOW){
        stopConveyor();
        stopMoving();
        //sideRangeMin = CLOSE_SIDE_RANGE_MIN;
        //sideRangeMax = CLOSE_SIDE_RANGE_MAX;

setArray(sideRearRangeArray, SIDE_ARRAY_SIZE, (SIDE_REAR_MIN+SIDE_REAR_MAX)/2);
    conveyorState = STOP;
    curY = 0;
    state = HOME;

    }else if(checkFrontRange() > FRONT_DIST){
        stopMoving();
        curY = 0;
        setArray(frontRangeArray, ARRAY_SIZE, 0);

        destination = SECOND_LEFT90;
        state = SECOND_LEFT;
    }
    break;

case (SECOND_LEFT):
    if(updateCanCount() == LOW){
        stopConveyor();
        stopMoving();
        conveyorState = STOP;
        state = REVERSE_TURN;
    }else if(curY < destination){
        turnLeft();
    }else{
        stopMoving();
        sideRangeMin = CLOSE_SIDE_RANGE_MIN;
        sideRangeMax = CLOSE_SIDE_RANGE_MAX;
        setArray(frontRangeArray, ARRAY_SIZE, 0);

setArray(sideRearRangeArray, SIDE_ARRAY_SIZE, (SIDE_REAR_MIN+SIDE_REAR_MAX)/2);

setArray(sideRangeArray, SIDE_ARRAY_SIZE, (CLOSE_SIDE_RANGE_MIN+CLOSE_SIDE_RANGE_MAX)/2);
    curY = 0;
    state = THIRD_STRAIGHT;
    }
    break;

case (THIRD_STRAIGHT):
    goForward(CORRECT);

    if(updateCanCount() == LOW){
        stopConveyor();
        stopMoving();
        conveyorState = STOP;
        //sideRangeMin = CLOSE_SIDE_RANGE_MIN;
        //sideRangeMax = CLOSE_SIDE_RANGE_MAX;

```

```

setArray(sideRearRangeArray,SIDE_ARRAY_SIZE,(SIDE_REAR_MIN+SIDE_REAR_MAX)/2);
    state = REVERSE_THIRD_STRAIGHT;
}else if(checkFrontRange() > FRONT_DIST){
    stopMoving();
    curY = 0;
    destination = LEFT180;
    setArray(frontRangeArray,ARRAY_SIZE,0);

setArray(sideRearRangeArray,SIDE_ARRAY_SIZE,(SECOND_SIDE_RANGE_MIN+SECOND_SIDE_RANGE_MAX)/2);
    state = THIRD_LEFT;
}
break;

case (THIRD_LEFT):
    if(curY < destination){
        turnLeft();

        if(conveyorState == FORWARD){
            if(updateCanCount() == LOW){
                stopConveyor();
                conveyorState = STOP;
            }
        }
    }else{
        stopMoving();
        curY = 0;
        sideRangeMin = SECOND_SIDE_RANGE_MIN;
        sideRangeMax = SECOND_SIDE_RANGE_MAX;
        setArray(frontRangeArray,ARRAY_SIZE,0);

setArray(sideRangeArray,SIDE_ARRAY_SIZE,(SECOND_SIDE_RANGE_MIN+SECOND_SIDE_RANGE_MAX)/2);
    state = FOURTH_STRAIGHT;
}
break;

case (FOURTH_STRAIGHT):
    goForward(CORRECT);
    if(conveyorState == FORWARD){
        if(updateCanCount() == LOW){
            stopConveyor();
            conveyorState = STOP;
        }
    }
    if(checkFrontRange() > FRONT_DIST){
        stopMoving();
        curY = 0;
        destination = LEFT90;
        setArray(frontRangeArray,ARRAY_SIZE,0);
        setArray(sideRangeArray,SIDE_ARRAY_SIZE,0);
        state = FOURTH_LEFT;

        //while(1);
    }
break;

```

```

case (FOURTH_LEFT):
    if(curY < destination){
        turnLeft();

        if(conveyorState == FORWARD){
            if(updateCanCount() == LOW){
                stopConveyor();
                stopMoving();
                conveyorState = STOP;
                state = HOME;
                //sideRangeMin = CLOSE_SIDE_RANGE_MIN;
                //sideRangeMax = CLOSE_SIDE_RANGE_MAX;

setArray(sideRearRangeArray, SIDE_ARRAY_SIZE, (SIDE_REAR_MIN+SIDE_REAR_MAX)/2);
            }
        }
    }else{
        stopMoving();
        curY = 0;
        //sideRangeMin = CLOSE_SIDE_RANGE_MIN;
        //sideRangeMax = CLOSE_SIDE_RANGE_MAX;

setArray(sideRangeArray, SIDE_ARRAY_SIZE, (CLOSE_SIDE_RANGE_MIN+CLOSE_SIDE_RANGE_MAX
)/2);
        state = HOME; //TODO case where we already went one loop and not collect
all cans
    }
    break;

case (REVERSE_THIRD_STRAIGHT):
    if(checkRearRange() == LOW){
        //overshoot reverse. Now too close to wall, move forward
        stopMoving();
        curY = 0;
        while(curY < 3410){
            goForward(CORRECT);
        }
        stopMoving();
        curY = LEFT90;
        state = REVERSE_TURN;

        setArray(rearRangeArray, ARRAY_SIZE, HIGH);

    }else if(curY > 0){
        goReverse();
    }else{
        stopMoving();
        curY = LEFT90;
        state = REVERSE_TURN;
    }
    break;

case (REVERSE_TURN):
    if(curY > 0){
        reverseTurnLeft();
    }else{
        stopMoving();

```

```

    curY = 0;
    //sideRangeMin = CLOSE_SIDE_RANGE_MIN;
    //sideRangeMax = CLOSE_SIDE_RANGE_MAX;

setArray(sideRearRangeArray, SIDE_ARRAY_SIZE, (SIDE_REAR_MIN+SIDE_REAR_MAX)/2);
    state = HOME;
}
break;
case (HOME):
    //Serial.print("HOME");
    goReverse();

    if(checkRearRange() == LOW){
        stopMoving();
        curY = 0;
        setArray(rearRangeArray, ARRAY_SIZE, HIGH);
        state = DROP_OFF;
    }
    break;

case (DROP_OFF):

    canCount++;
    //while(digitalRead(CAN_COUNT_SENSOR) == LOW){
        curY = 0;
        openCanCompartment();

        setArray(canCountArray, CAN_ARRAY_SIZE, HIGH);
        while(curY < 1500){
            goForward(NO_CORRECT);
        }
        closeCanCompartment();
        stopMoving();
    //}
    //maybe do something different
    if(canCount == 1){
        sideRearMin = SIDE_REAR_MIN2;
        sideRearMax = SIDE_REAR_MAX2;
        startConveyor();
        conveyorState = FORWARD;
        state = SECOND_STRAIGHT;
        setArray(frontRangeArray, ARRAY_SIZE, 0);
    }else{
        stopConveyor();
        curY = 0;
        while(curY > -500)
            goReverse();
        stopMoving();
        curY = 0;
        state = PROTECT;
    }
    break;

case (PROTECT):
    if(curY < -200)
        goForward(NO_CORRECT);
    else if(curY > 200)

```

```

        goReverse();
    else
        stopMoving();

    break;

default:
    state = PROTECT;
    break;
};
}

int checkSideRange() {
    int sensorVal = analogRead(SIDE_RANGE_PIN);

    sideRangeArray[index3%SIDE_ARRAY_SIZE] = sensorVal;
    index3++;
    Serial.print(sensorVal);
    Serial.println();
    return getMedian(sideRangeArray, SIDE_ARRAY_SIZE);
}

int checkSideRearRange() {
    int sensorValue = analogRead(SIDE_REAR_PIN);

    sideRearRangeArray[index4%SIDE_ARRAY_SIZE] = sensorValue;
    index4++;
    Serial.print(sensorValue);
    Serial.println();
    return getMedian(sideRearRangeArray, SIDE_ARRAY_SIZE);
}

int checkFrontRange() {
    int sensorVal = analogRead(FRONT_RANGE_PIN);

    frontRangeArray[index%ARRAY_SIZE] = sensorVal;
    //Serial.print(sensorVal);
    //Serial.println();
    index++;
    return getMedian(frontRangeArray, ARRAY_SIZE);
}

int checkRearRange() {
    if(digitalRead(REAR_RANGE_SENSOR) == LOW)
        rearRangeArray[index2%ARRAY_SIZE] = LOW;
    else
        rearRangeArray[index2%ARRAY_SIZE] = HIGH;

    index2++;
    return getMedian(rearRangeArray, ARRAY_SIZE);
}

void setArray(int *a, int arraySize, int val) {
    for(int i = 0; i < arraySize; i++) {
        a[i] = val;
    }
}

int getMedian(int *a, int n) {

```

```

int midpoint = n/2;

for (int i = 1; i < n; ++i){
    int j = a[i];
    int k;
    for (k = i - 1; (k >= 0) && (j < a[k]); k--){
        a[k + 1] = a[k];
    }
    a[k + 1] = j;
}

return a[midpoint];          //return the median of the array
}

void goForward(int mode){
    //updateMouse();
    int curSideRange;

    if(mode == CORRECT){
        curSideRange = checkSideRange();
        if((state == THIRD_STRAIGHT) && (curSideRange < 200)){
            if((curY % 1000) > 850){
                MyServo[L_DRIVE].write(v[L_DRIVE][FORWARD]);
                MyServo[R_DRIVE].write(v[R_DRIVE][STOP-3]);
            }else{
                MyServo[L_DRIVE].write(v[L_DRIVE][FORWARD]);
                MyServo[R_DRIVE].write(v[R_DRIVE][FORWARD]);
            }
        }else if(curSideRange > sideRangeMax){
            MyServo[L_DRIVE].write(v[L_DRIVE][STOP+3]);
            MyServo[R_DRIVE].write(v[R_DRIVE][FORWARD]);
        }else if(curSideRange < sideRangeMin){
            MyServo[L_DRIVE].write(v[L_DRIVE][FORWARD]);
            MyServo[R_DRIVE].write(v[R_DRIVE][STOP-3]);
        }else{
            MyServo[L_DRIVE].write(v[L_DRIVE][FORWARD]);
            MyServo[R_DRIVE].write(v[R_DRIVE][FORWARD]);
        }
    }else{
        MyServo[L_DRIVE].write(v[L_DRIVE][FORWARD]);
        MyServo[R_DRIVE].write(v[R_DRIVE][FORWARD]);
    }
    //delay(2);
    updateMouse();
}

void goReverse(){
    int curSideRearRange = checkSideRearRange();

    if(curSideRearRange > sideRearMax){
        MyServo[L_DRIVE].writeMicroseconds(LEFT_SLOW_REV);
        MyServo[R_DRIVE].write(v[R_DRIVE][REVERSE]);
    }else if(curSideRearRange < sideRearMin){
        MyServo[L_DRIVE].write(v[L_DRIVE][REVERSE]);
        MyServo[R_DRIVE].writeMicroseconds(RIGHT_SLOW_REV);
    }else{
        MyServo[L_DRIVE].write(v[L_DRIVE][REVERSE]);
        MyServo[R_DRIVE].write(v[R_DRIVE][REVERSE]);
    }
}

```

```

    }
    updateMouse();
}

void stopMoving() {
    MyServo[L_DRIVE].write(v[L_DRIVE][STOP]);
    MyServo[R_DRIVE].write(v[R_DRIVE][STOP]);
    updateMouse();
}

void turnLeft() {
    MyServo[L_DRIVE].write(v[L_DRIVE][STOP]);
    MyServo[R_DRIVE].write(v[R_DRIVE][FORWARD]);
    updateMouse();
}

void reverseTurnLeft() {
    MyServo[L_DRIVE].write(v[L_DRIVE][STOP]);
    MyServo[R_DRIVE].write(v[R_DRIVE][REVERSE]);
    updateMouse();
}

void turnRight() {
    MyServo[L_DRIVE].write(v[L_DRIVE][FORWARD]);
    MyServo[R_DRIVE].write(v[R_DRIVE][STOP]);
    updateMouse();
}

void startConveyor() {
    digitalWrite(SWEEPERS, HIGH);
    MyServo[B_CONVEYOR].write(v[B_CONVEYOR][FORWARD]);
    MyServo[T_L_CONVEYOR].write(v[T_L_CONVEYOR][FORWARD]);
    MyServo[T_R_CONVEYOR].write(v[T_R_CONVEYOR][FORWARD]);
}

void stopConveyor() {
    digitalWrite(SWEEPERS, LOW);
    delay(100);
    MyServo[B_CONVEYOR].write(v[B_CONVEYOR][STOP]);
    MyServo[T_L_CONVEYOR].write(v[T_L_CONVEYOR][STOP]);
    MyServo[T_R_CONVEYOR].write(v[T_R_CONVEYOR][STOP]);
}

void openCanCompartment() {
    MyServo[TRAP_DOOR].write(v[TRAP_DOOR][FORWARD]);

    for(int i = 0; i < 10; i++){
        updateMouse();
        delay(20);
    }
    MyServo[SLIDE_DOOR].write(v[SLIDE_DOOR][FORWARD]);

    for(int i = 0; i < 100; i++){
        updateMouse();
        delay(20);
    }
    MyServo[SLIDE_DOOR].write(v[SLIDE_DOOR][STOP]);
}

```

```

    for(int i = 0; i < 5; i++){
        updateMouse();
        delay(20);
    }
}

void closeCanCompartment() {
    MyServo[SLIDE_DOOR].write(v[SLIDE_DOOR][REVERSE]);
    MyServo[TRAP_DOOR].write(v[TRAP_DOOR][REVERSE]);

    for(int i = 0; i < 60; i++){
        updateMouse();
        delay(20);
    }
    MyServo[SLIDE_DOOR].write(v[SLIDE_DOOR][STOP]);
}

int updateCanCount() {
    previousState = digitalRead(CAN_COUNT_SENSOR);
    delay(5);

    if(digitalRead(CAN_COUNT_SENSOR) == LOW && previousState == LOW) {
        canCountArray[index5%CAN_ARRAY_SIZE] = LOW;
    } else {
        canCountArray[index5%CAN_ARRAY_SIZE] = HIGH;
    }

    index5++;
    return getMedian(canCountArray, CAN_ARRAY_SIZE);
}

void mouse_init()
{
    mouse.write(0xff); // reset
    mouse.read(); // ack byte
    mouse.read(); // blank */
    mouse.read(); // blank */
    mouse.write(0xf0); // remote mode
    mouse.read(); // ack
    delayMicroseconds(100);
}

void updateMouse() {
    char mstat;
    char mx;
    char my;

    /* get a reading from the mouse */
    mouse.write(0xeb); // give me data!
    mouse.read(); // ignore ack
    mstat = mouse.read();
    mx = mouse.read();
    my = mouse.read();
    //curX += (mx/10);
    curY += (my/2);
}

```