Abstract

The project of the course required us to move the Mechbot along a black taped line while communicating with the EVShield. Large amounts of iterations allowed the Mechbot to consistently stay along the path of the black tape and stop at locations where there were double tape black lines on the floor. The EVShield would receive a high bit from the Mechbot and actuate motors to rotate so a specific colored can could be picked and delivered to a desired location. The EVShield arm required a robust structure so that it did not fall apart every time motors were activated by the EVShield. Simulink models were made based on the entire system of the project to create virtual models of how the arm performed when gripping and releasing the cans. It was recommended that it would have been better for the experience of all students if more sensors were used with the EVShield instead of just the camera. The project is expected to perform well at the time of the presentation of the project demo.

Table of Contents

1.0 Introduction

The project is conducted to test the Mechbot and EVShield on a track with obstacles such that the Mechbot is able to move along the black line path while communicating with the EVShield. The entire program in the Mechbot and the EVShield are both made in Arduino IDE but with significant differences in the directory files that are used for their control. The Mechbot is programmed to ensure that it reads the black line and follows it throughout the duration of the Project. On approaching a can on one of the two sides of the Mechbot, the Mechbot is programmed to stop and instruct the EVShield to move its motor to the left or right direction, pick the can from it position and continue motion along the black line while still gripping onto the can. The EVShield consists of three motors; one motor to allow the arm to move vertically upwards and downwards, one motor to move horizontally left and right and the last motor to grip and release the can it holds. The Mechbot, after picking a can is programmed to continue its travel on the black line to position where the Mechbot must stop and the arm must release the can. The detection of the can and its position of release on the pathway is detected by the camera that is attached to the EVShield. All the motors and the camera are connected to the EVShield, The EVShield is then in turn connected to the Mechbot. Connection between the EVShield and the Mechbot is done via pins.

2.0 Literature Review

The line sensors at the bottom of the Mechbot are the significant factors that detects the dark line on the floor. Based on conditions created on the program and the data received from the black line on the ground, the motors 1 and motor 2 on the Mechbot are turned on to move in a particular direction. The line sensors on the Mechbot used at the Project detects high values when presented to black line and shows very low values when presented with a white line. There are four line sensors at the bottom, with the left most sensor called analog 0 and the right most line sensor called analog 3. The lowest value the line sensors are able to detect is 0 and the highest value is 1000. When the motor reaches a double taped black line, it is also coincidentally the same position where it is expected that a can is positioned to the left or right of the Mechbot or it also coincidentally the same place of releasing the can to the left or right of the Mechbot. Pins C4 and Pins C5 on the Mechbot are used to direct information towards the EVShield while Pin 0 and Pin 1 on the EVShield are used to direct information towards the Mechbot. The arm attached to the Mechbot consists of three motors, they rotate at fixed degrees set in the program. At first, the first motor simply moves vertically downwards to be at the correct height to pick the can up, the second motor is used to move the arm 90 degrees to the left or 90 degrees to the right horizontally from the center to directly face the gripping arm towards the can. The third motor is finally on the gripping arm, to get a hold on the can. After a gripping hold is made on the can, the arm moves to the center and the Mechbot continue its motion towards the releasing position of the can where it follows the same procedure as it does when picking up the can but this time it will release the can. The speed of the motors on the arm were programmed to be slow to prevent hitting the can before the can could be gripped. On the pathway a garbage disturbance is placed which is detected by any of the front bumpers which stops the Mechbot and waits for the garbage disturbance to be removed before the Mechbot continues to complete its path.

3.0 Problem Definition

The aim of this project is to create a prototype automated guided vehicle (AGV) to perform part delivery tasks in an industrial setting. It must be guided using black lines on the floor and pick up and drop off cans at two types of receptacles based on can colour. It must also recognise and stop its motion when obstacles are in its path. A complete list of product characteristics and functional requirements are presented in Table 1.

The AGV must operate in an industrial environment at all ambient conditions found therein. This may include reduced air quality, temperature variations of 10-20 degrees Celsius from ambient, and all levels of humidity. Users of this product include material handling personnel that work in the vicinity of the AGV and maintenance personnel that repair and maintain the equipment. At this prototype stage our design is not concerned with operating conditions or specific user groups at this proof-of-concept stage however our team is aware of them and further iterations will continue to be designed around them.

Table 1. Product characteristics, functional requirements, constraints, metrics

Product Characteristics	Functional Requirements	Constraints	Performance Metric
Based on Mechbot and Lego	Can only use Mechbot 2.5, EV Shield, Lego mindstorms kit	No other components used	P=100*(# of other components/# of other components)
Mobile	Move under its own power along the track at reasonable speed	V<1mm/s	P=100*(Vmin-V)/Vmin
Retrieval system	Must pick up and carry cans	Pick up cans from either side and deposit in center	P=50*(# of sides serviced)
Decisive	Must drop cans off at correct destinations	Only red cans on left side receptacles, blue cans on right side receptacles	P=100*[(# of red cans on left)+(# of blue cans on right)]/ [(# of red cans on left)+(# of blue cans on right)]

Self-navigating	Must detect and follow black tape path	Does not deviate from black line for more than 0.1s	P=100*(0.1-Tmax)/0.1
Self-stopping	Must stop at stations marked with cross tape	Does not miss any station	P=100*(# of missed stations)/(# of missed stations)
Safe to use around obstacles	Must stop when obstacles present	Does not miss any	P=100*(# of missed obstacles/# of missed obstacles)
Consistent	Delivered cans must remain at point	No cans leave the receptacles	P=(# cans that leave the receptacle)

3.1 Black Box Diagram

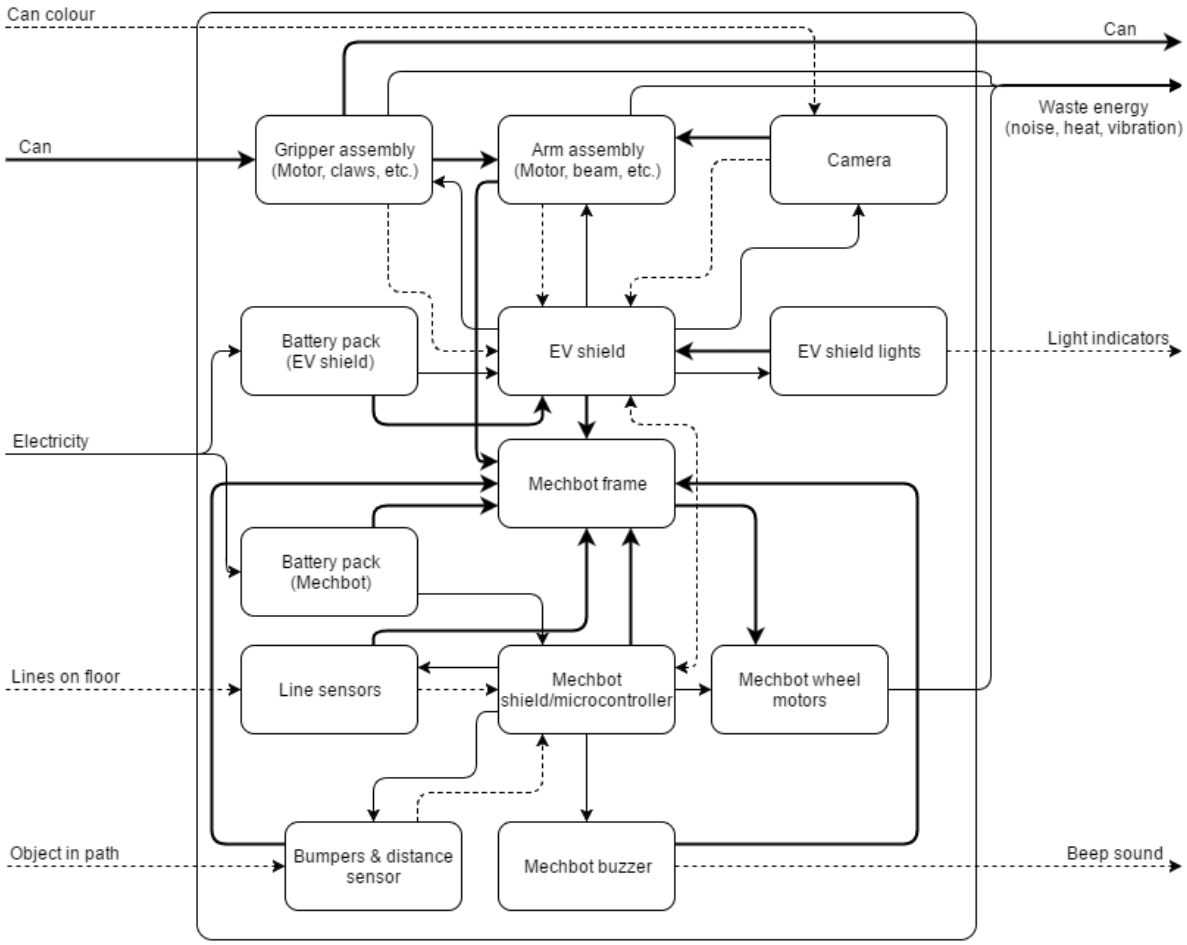

Figure 1. Black box diagram of required subsystems

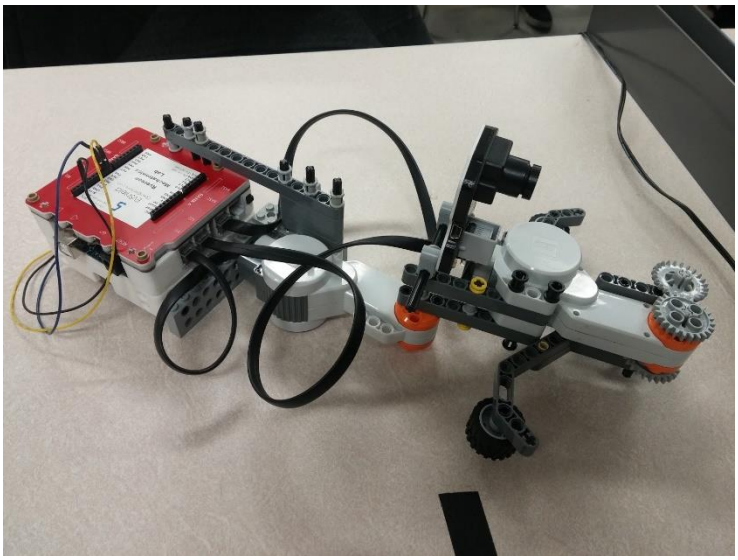

Figure 2. Lego EV Shield arm assembly

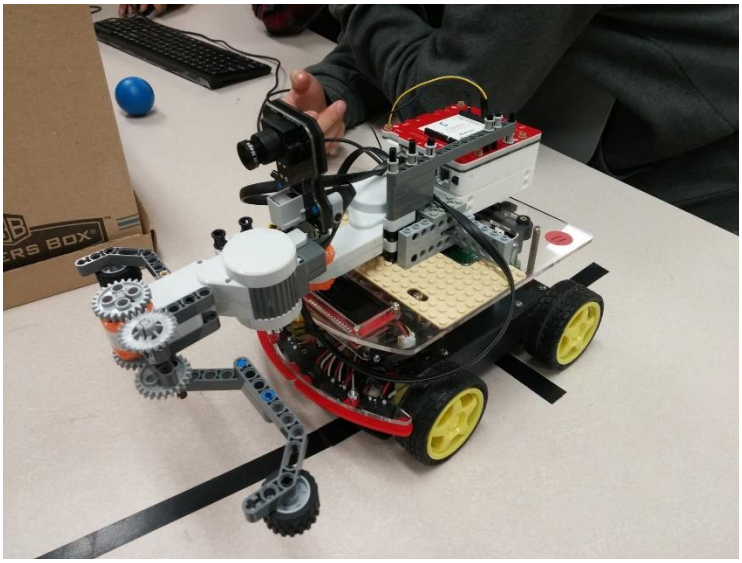

Figure 3. Complete AGV prototype including Mechbot 2.5

4.0 Design Solutions

Our final design is the result of many iterations of both programming and mechanism designs to satisfy all of the functional requirements.

Our team decided very early in the design process that the Mechbot with its mounted gear motors already connected to wheels would be the simplest way to satisfy the mobility requirements. It also has conveniently located bumper switches and distance sensors to detect objects in its path and floor-facing line sensors to follow the line and detect station markers. This decision proved obvious and no changes were deemed necessary.

Because the Lego sensors and motors require programming with a unique pseudo-language and we were provided with a cable connecting the RX and TX pins of the Mechbot embedded microcontroller and the EV Shield it was deemed much easier to separate the AGV mobility functions from the functions interacting with the can between the Mechbot embedded microcontroller and the EV Shield respectively. It was considered whether to utilise I2C serial protocol to communicate between the two microcontrollers but as less than four different instructions were required to be communicated between the microcontrollers it could be done much more computationally efficient by controlling the state of the two connected pins in different combinations to communicate instructions.

Our arm design is based on the team's assessment of the problem of the cans falling over and rolling away when deposited at the station. A design utilising a third motor to raise and lower the grabber through a parallelogram four-bar mechanism was built and tested to address this issue. After much trial-and-error it was decided to abandon this approach due to structural instability of the arm and the length that this motor added to the arm would over reach the can pick up point and we had no other motor to solve this problem by collapsing the arm is some way. A simpler solution using a much simpler arm design shown in Figure 1 was devised where the grabber would open only a few degrees when releasing the can so that the can would be stabilised by the grabber arms as it falls. Once settled on the receptacle the grabber would open completely to permit free rotation of the arm without contacting the can.

5.0 SimMechanics Model

Simulink was used to create SimMechanics as representation of our robot. Dimensions were taken of robotic arm to be used in our SimMechanics.

The values used to represent the motor that revolves the arms are:

Kb= 0.46839 (V/(rads/s))

Kt= 0.31739 (N.m/A)

The following are the calculations that were completed for each body part. We had to calculate the inertia using the dimensions measured.

5.1 Calculations for SimMechanics

Inertia tensor:

Body:

Dimensions

m=18g

x= 12.7cm

y= 3.6 cm

z= 0.8 cm

$I_1 = (m/12)(b^2+c^2) = 20.4 \text{ g x cm}^2$

$I_2 = (m/12)(a^2+c^2) = 242.895 \text{ g x cm}^2$

$I_3 = (m/12)(a^2+b^2) = 361.375 \text{ g x cm}^2$

Body 1:

Dimensions

m=1.89g

x= 0.2519 cm

y= 4 cm

z= 0.378 cm

$I_1 = (m/12)(b^2+c^2) = 2.5425$ g x cm^2

$I_2 = (m/12)(a^2+c^2) = 0.03249$ g x cm^2

$I_3 = (m/12)(a^2+b^2) = 2.529$ g x cm^2

Body 2,3,5,6,7,9 and 10

Dimensions

m=6g

x= 12.7 cm

y= 1.2 cm

z= 0.8 cm

$I_1 = (m/12)(b^2+c^2) = 1.04$ g x cm^2

$I_2 = (m/12)(a^2+c^2) = 80.9$ g x cm^2

$I_3 = (m/12)(a^2+b^2) = 81.4$ g x cm^2

Body 11 and 12

Dimensions

m=1.5g

radius= 1.5875 cm

h= 1.2 cm

$I_1 = (m/4)(R^2+h^2/3) = 1.125$ g x cm^2

$I_2 = (m/4)(R^2+h^2/3) = 1.125$ g x cm^2

$I_3 = m\ R^2/2 = 1.89$ g x cm^2

The following figures illustrate our Simulink program and the graphs that are created to demonstrate the motion of our robotic arm.

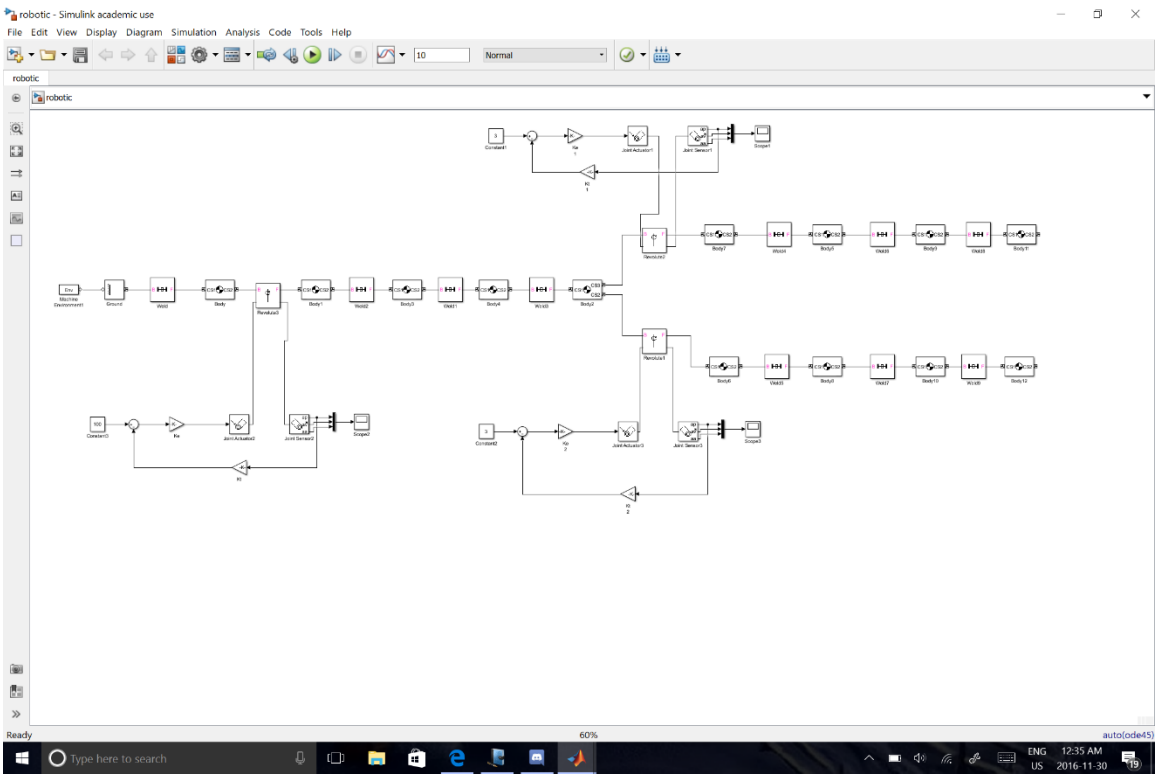

Figure 4: Simulink program

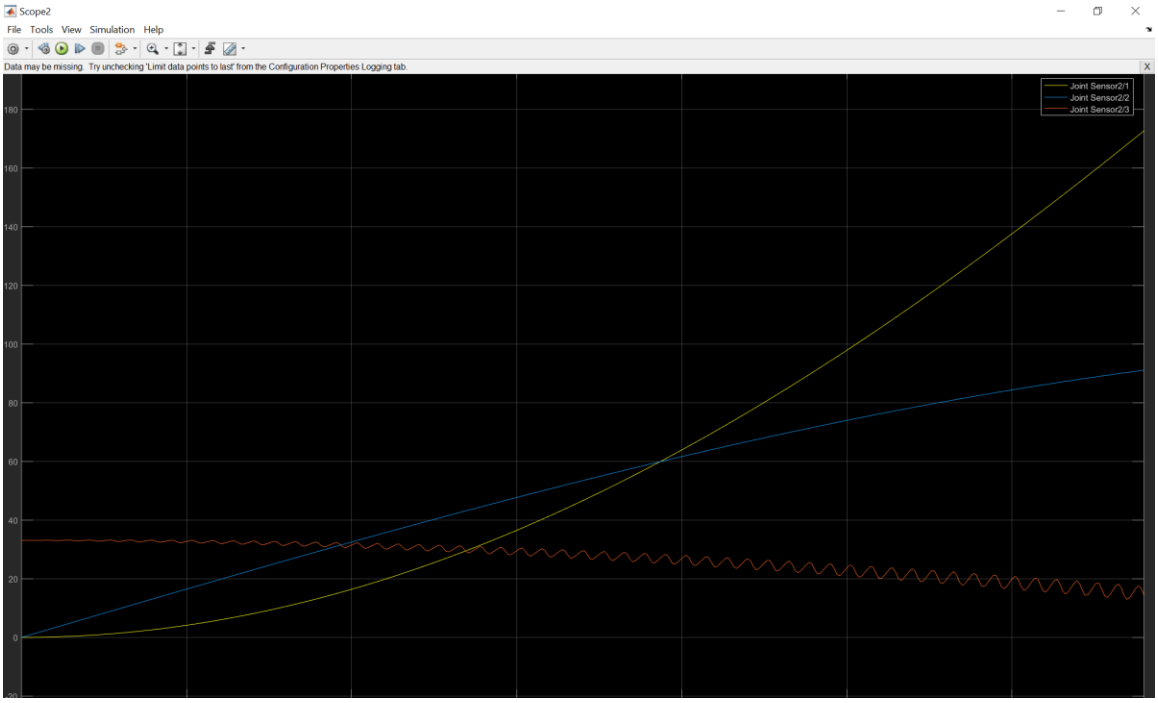

Figure 5: Graph representing the motion of the robotic arm

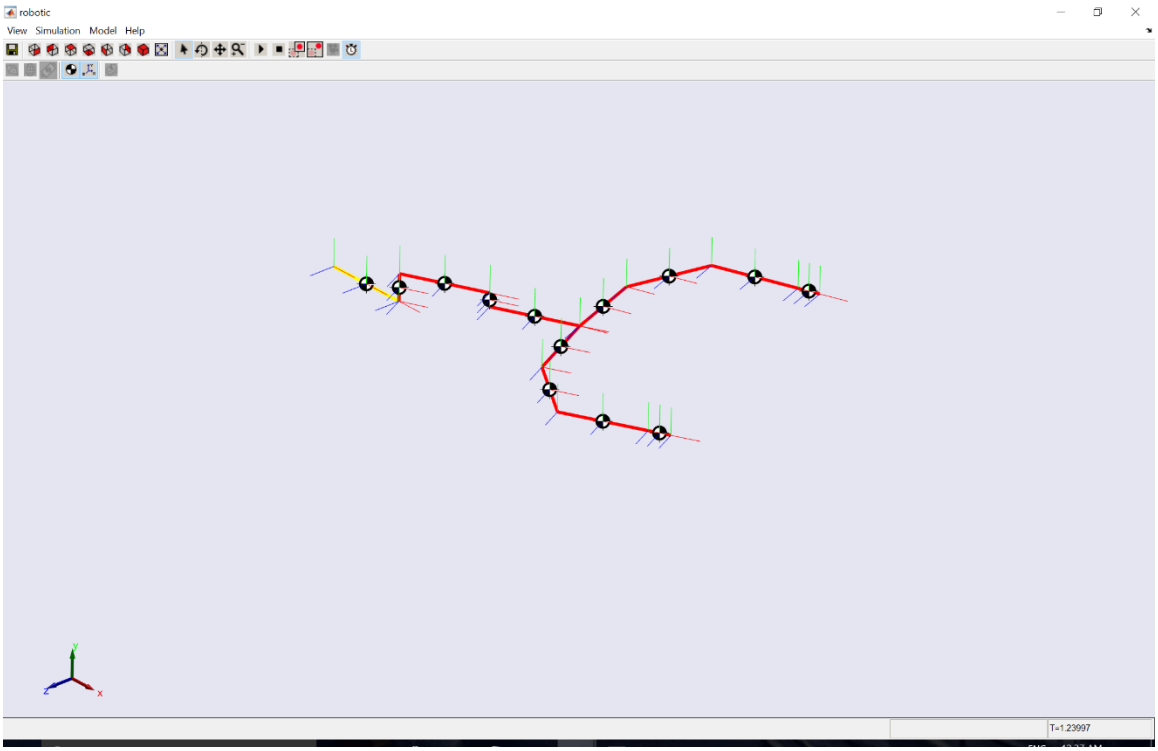

Figure 6: SimMechanics of the robotic arm

6.0 Discussion of Systems

6.1 Mechanical System

In this project, 2 Lego motors and an assortment of Lego blocks from a Lego Mindstorms NXT build kit were used to develop a fully functional mechanical arm able to securely pick up empty pop cans and deliver them to specified locations. The mechanical build of this system had to be structurally-sound enough to support not only the weights of the motors and pop cans, but also the loads of the EVShield, Lego NXT camera, and all the supporting pieces of Lego used.

Ultimately, a satisfactory design was developed that met all of these requirements. In the design, the EVShield acted as a counterweight to the front-loaded robot arm mounted on top of the Mechbot. Both motors were implemented in such a way to provide the necessary range of motion to complete the challenge, while adhering to the constraints set out in the original project or those of which that were realized during the progression of the project. One motor controlled the lateral movement of the arm, swinging it between the Mechbot's left side to its right side. The other motor was used to control the operation of the grabbing claw. By using the gearing system at the hinge of the claw, a sufficient force and speed was able to be developed to consistently maneuver the pop can along its course. Analog sensors such as the black line reading and proximity sensors were also utilized in this project, in order to more-accurately check for the correct conditions of any situation. Outputs such as the LED bar and buzzer on the Mechbot and the LEDs on the EVShield were used to provide feedback back to the developers, testers, and examiners of this system in order to improve its user interface, and to also meet any formal requirements expected of this project.

6.2 Electrical System

The Mechbot and the EVShield were interfaced using a method based on the I^2C serial communication protocol, via a connecting cable. Using this cable, two selected pins in the Mechbot were able to transfer data with two selected pins on the EVShield. These pins were simply used to pull their signals high or low, which corresponded to certain actions defined in the software of this system. In this project, pins PC4 and PC5 were selected on the Mechbot to communicate with the pins PB0 and PB1 on the EVShield, respectively. Other combinations may have worked as well, but a pre-existing familiarity with this specific pin combination was already established prior to this project. Other than this serial communication method, other electrical components of this project included the battery pack on-board the Mechbot and EVShield, and the actual wiring connecting all the Arduino boards to the Mechbot and EVShield. These components, however, are not expected or intended to be manipulated in order to complete this project, and likely would not be recommended. The actual electrical wiring and

battery packaging are not within the scope of this course, so any troubleshooting regarding these components would be performed by the assistance of the lab technicians and staff.

There were two programs used to complete this project: one to control the Mechbot and send appropriate signals to the EVShield; and the other to control the EVShield and NXT camera system and send appropriate signals back to the Mechbot.

6.3 Software System

In the Mechbot program, the mechbotShield and avr/io header files were imported. The main program was then entered, where most of the significant work was completed. All necessary motors, LEDs, buzzers, bumpers, line sensors, and distance sensors were initiated. The pins for the bumpers, PC4 and PC5 were initially set low for input. If the pins are pulled high, a signal will be sent to the EVShield program to perform the appropriate actions. PC4 will be pulled high if the Mechbot stops at one of the stations. A signal will then be sent to the EVShield to move the robot arm system to manipulate the pop can in an appropriate manner. Otherwise, if none of the bumper switches, PC4, or PC5 are pulled high, the Mechbot will perform its main line following procedure. In the line following procedure, the line sensors are continuously polled in order to determine the Mechbot's position with respect to the black line while it is moving forward. Depending on its position, the Mechbot will adjust left or right to ensure that it is always located on the line. Although a PID control system would likely help the Mechbot navigate on the track smoothly, a simple polling-and-adjusting method was determined to be sufficient for the purposes of this project. Once all four line sensors detect a dark line, the Mechbot stops, and pulls PC4 high to initiate the EVShield program. If the Mechbot detects garbage on the track via its bumpers or front distance sensor, it will stop and beep its buzzers. PC5 will then pull high and this change should register in the EVShield program to flash its lights red. This process will keep looping until the Mechbot reaches the end of the track.

In the EVShield program, the Wire, EVShield, EVs_NXTCam, USART, mechbotShield, and avr/io header files were all imported. The EVShield and NXT camera was initialized. Then the camera's accompanying variables nblobs, color[i], and i were also initialized. Two counter variables were initiated and set as LS=0 and RS=0. Afterwards, the setup procedure was run to initialize and reset all the motors, setup the I2C hardware capabilities, and setup the tracking ability of the NXT camera. The appropriate USART pins were then set as outputs. For this project, PB0 and PB1 were initially set low. These pins were wired with pins PC4 and PC5, respectively, on the Mechbot interface. If the Mechbot sent an input signal to the EVShield, the corresponding pin should pull high and perform some action.

In the loop procedure, the NXT camera was activated to setup its field view. Then, a while loop checks to see if PB0 is pulled high. If it is pulled high, it indicates that the Mechbot is stopped at a station. Here, the program determines which station it is on: a pickup station or a drop-off station. The values of LS and RS are checked in order to do this – if both LS and RS equal 0, then the EVShield will command the robot arm to try to pick up a pop can. If either LS or RS equal 1, then the robot arm will perform its drop-off actions. At a pickup station, the robot arm will move to its right to check for a can. It will first check for a red can. If a red can exists, the arm will close its claws and move it to in front of the Mechbot. If a red can is grabbed, the counter LS is set to 1. If it does not detect a red can, it will check for a blue can. If blue is detected, it will close its claws and move the can to in front of itself. If a blue can is grabbed, the counter RS is set to 1. If neither colour is detected, the robot arm will move to the left side of the Mechbot and check for a can similar to the method used for the right side. At this point, a can should be detected and moved to the front and either LS or RS should equal to 1. Once a can is grabbed, the pin PB0 should be set low to exit the EVShield procedures and get back to the Mechbot program. The Mechbot is then expected to move along its track until it reaches another stop. At a drop-off location, the robot arm will move to either the left of right side depending on whether LS or RS equals 1. After moving to the appropriate side, the claw will open to release the can onto the correct-coloured platform. The counter LS or RS will then be set back to equal 0, to get ready for the next station. This process is repeated until the Mechbot reaches the end of the test track. At each stop, PB1 should be set high to signal the LEDs of the EVShield to light red. Otherwise, if PB1 is set low (default), the EVShield is lit green to indicate that the Mechbot is moving.

Flowchart of Mechbot

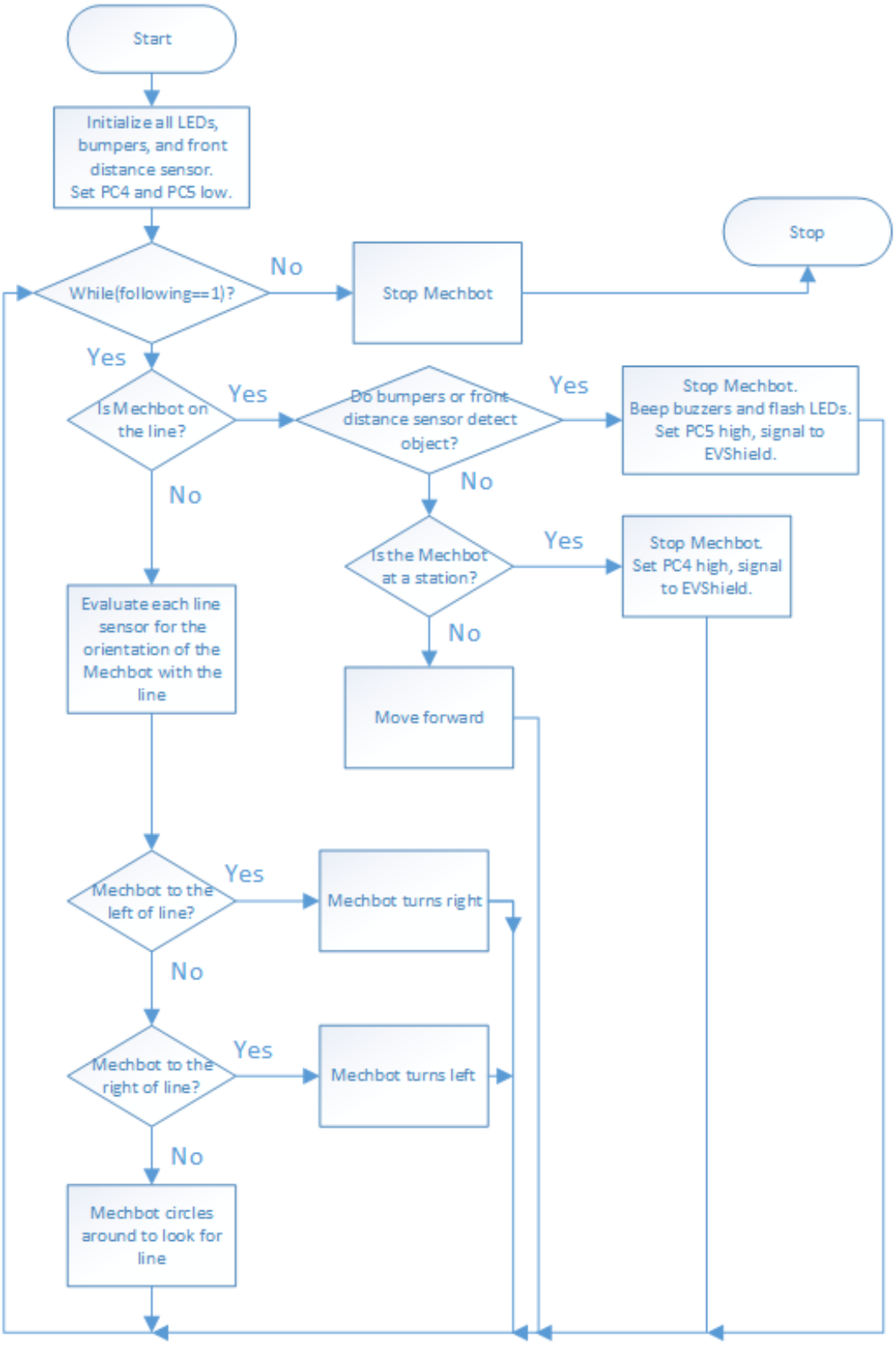

Figure 7: Flowchart of the Mechbot program

Flowchart of EVShield

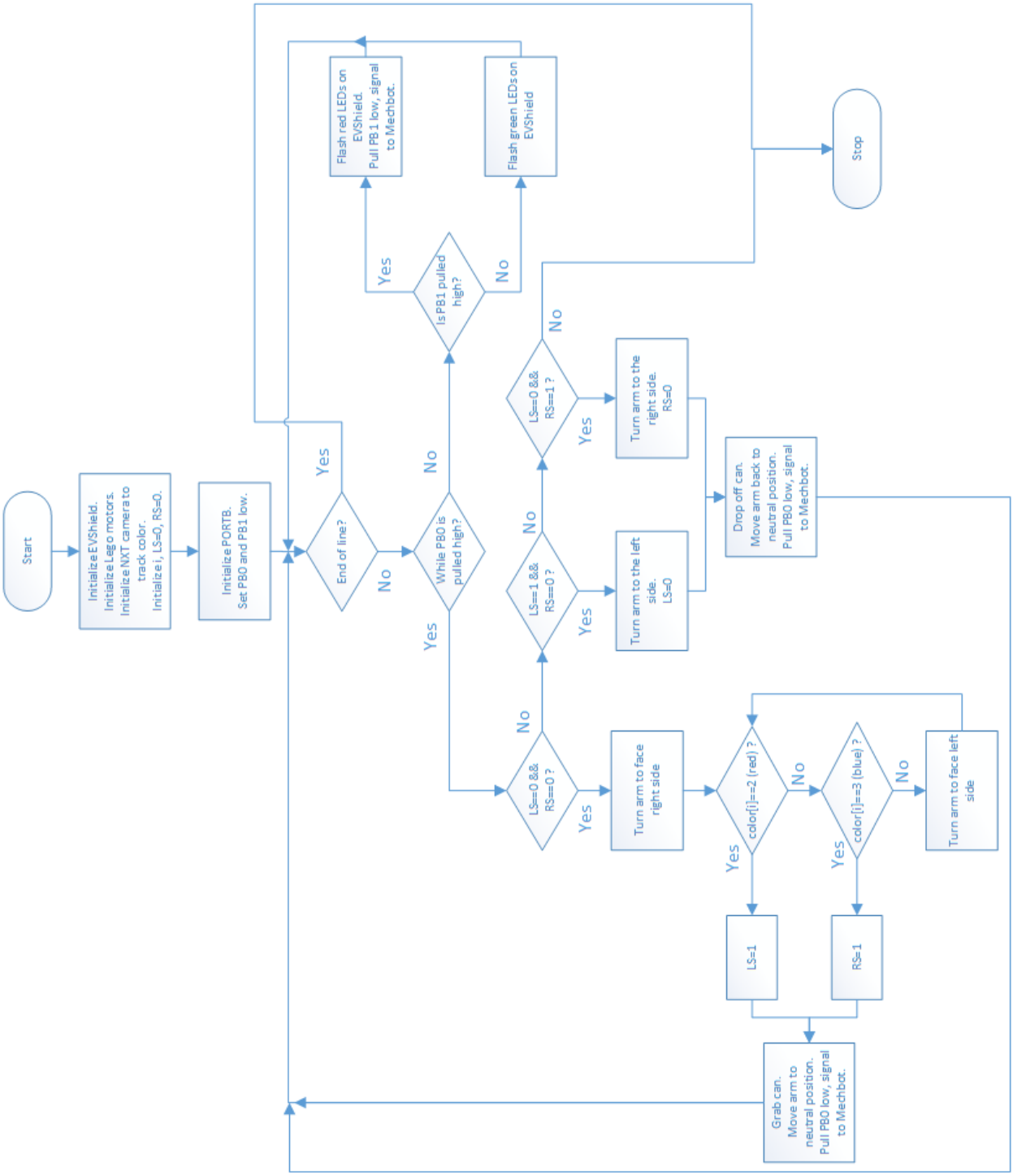

Figure 8: Flowchart of the EVShield program

7.0 Team Roles

Kirollos El - Masry	Created the Simulink and SimMechanics model for the robot arm design. Calculated all required dimensions and relevant numerical information. Assisted in the mechanical design.
Wilson Chu	Discussion of mechanical, electrical, and software systems. Flowcharts for all programs. Table of contents. Compiled and formatted the Project Report. Assisted in the development of the Mechbot and EVShield programs using Arduino IDE.
Faisal Uddin Mahboob	Created the program in Arduino IDE for the Mechbot's correct motion and EVShield arms rotation to pick and drop the colored cans. Perform Design optimization and maneuver system between constraints. Write the Introduction, Literature Review, Conclusion and Recommendations of the Project Report.
Zachary Hickey	Mechanical design, mechanical implementation, software implementation, problem definition, black box model, design iterations. Assisted in the development of the programs.

7.1 Technical Issues

Our team had repeated problems with the camera recognising the colour of the can and at times even the presence of a can, even with programs such as our lab 5 program that we knew was programmed correctly. This resulted in the AGV running the same program under the same input conditions exhibiting unpredictable colour recognition. The Mechbot designated to our group for testing had an over cycled battery that no longer held a useful charge, resulting in inconsistent and unpredictable behaviour when testing as the battery quickly drained.

8.0 Conclusions

The Mechbot and EVShield together performed the outlines of the project as was intended. The Mechbot continued to stop exactly on the location where there were double black tapes placed on the floor. It allowed the EVShield to perform the actions necessary to move the arm left or right to pick the necessary coloured can and bring it to the centre while still holding on to it. The Mechbot successfully placed the can at its designated locations. It stopped at the moment of impact with the garbage barrier and continued its normal motion when the garbage was removed. The Mechbot arms were changed at different parts of the project to ensure smooth motion of the arm. Initially, 3 motors were used in the movement of the arm. However, towards to end of the project it was decided to use only 2 motors. Since the third motor caused unnecessary extra motions as the cans were always accessible enough with just two motors. The third motor also made it difficult to hold together all the Lego pieces of the arm. In the beginning of the project the Mechbot continuously always moved at a high speed, the team members decided to reduce the speed to a much slower amount to ensure the Mechbot did not speed away from the black taped lines. In the end the Mechbot reached it final destination location as it was intended.

9.0 Recommendations

The Mechbot could have additional space for battery. This is because as the Mechbot was continually being practised on during the course, would run out of battery charge quite frequently. This would hinder the progress of the project and take away valuable time that could be used to perfect the end results. The EVShield happened to show the same problem, the only way the team continued to progress without the simple problems of battery charge was to keep an extra battery cartridge of the EVShield with its Lego pieces through the entire course. In the end it saved time and allowed more work to be done on the program generation. The project was almost entirely conducted on the camera sensor alone in terms of sensors being used by the EVShield. In the project, this was to detect the colored cans as the Mechbot was in motion. However, there were multiple other sensors available that could have been used, such as Light sensors, Touch sensors, Sound Sensors. These were sensors that would greatly broaden the exposure of the students to apply various sensors to an autonomous vehicle remotely. It would make the course extremely interesting and make the students more prepared to create robots after completing University. The project itself can be converted into one that can run parallel to a Capstone project that may span the entire school year instead being conducted in just one semester. This would allow ample time to introduce far more complications in the path for the Mechbot to travel on and will allow the students to use more different sensors of the EVShield. Student would become more enthusiastic about project deliverables and gain more necessary skills at being a Mechatronics Engineer after completing the Undergraduate degree.

10.0 Appendix

Mechbot Program

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

int main (void) {
 //intiate motor subroutine
 initMotor();
 //initiate ADC subroutine
 initADC();
 //set ddr for serial pin low for input
 DDRC &= ~(1 << PC4);
 DDRC &= ~(1 << PC5);
 //set ddrs low for input from bumpers
 DDRD &= ~((1 << PD3) | (1 << PD4) | (1 << PD5));
 DDRC &= ~(1 << PC1);
 //create variables
 int following = 1;
 int crit = 750;
 int power = 600;
 int quota = 0;
 //line following loop
 while (following == 1) {
   //condition for no line seen
   if ((analog(0) < crit) && (analog(1) < crit) && (analog(2) < crit) &&
(analog(3) < crit)) {
     //turn around to look for line
     motor(power, -power);
     if (quota = 3) {
       void playShaveAndHaircut (void);
     }
     delay_ms(100);
   }
   //condition for mechbot in center
   if ((analog(0) < crit) && (analog(1) > crit) && (analog(2) > crit) &&
(analog(3) < crit)) {
     //run straight
     motor(power, power);
     delay_ms(100);
   }
   //condition for mechbot slightly right
   if ((analog(0) > crit) && (analog(1) > crit) && (analog(2) > crit) &&
(analog(3) < crit)) {
     //turn slightly left
     motor(power / 2, power);
     delay_ms(100);
   }
   //condition for mechbot right
   if ((analog(0) > crit) && (analog(1) > crit) && (analog(2) < crit) &&
(analog(3) < crit)) {
     //turn left
```

```
    motor(0, power);
    delay_ms(100);
  }
  //condition for mechbot far right
  if ((analog(0) > crit) && (analog(1) < crit) && (analog(2) < crit) &&
(analog(3) < crit)) {
    //turn hard left
    motor(-power, power);
    delay_ms(100);
  }
  //condition for mechbot slightly left
  if ((analog(0) < crit) && (analog(1) > crit) && (analog(2) > crit) &&
(analog(3) > crit)) {
    //turn slight right
    motor(power, power / 2);
    delay_ms(100);
  }
  //condition for mechbot left
  if ((analog(0) < crit) && (analog(1) < crit) && (analog(2) > crit) &&
(analog(3) > crit)) {
    //turn right
    motor(power, 0);
    delay_ms(100);
  }
  //condition for mechbot far left
  if ((analog(0) < crit) && (analog(1) < crit) && (analog(2) < crit) &&
(analog(3) > crit)) {
    //turn hard right
    motor(power, -power);
    delay_ms(100);
  }
  //condition for stations
  if ((analog(0) > crit) && (analog(1) > crit) && (analog(2) > crit) &&
(analog(3) > crit)) {
    //stop
    //delay_ms(250);
    motor(-power, -power);
    delay_ms(100);
    motor(0, 0);
    delay_ms(250);
    PORTC |= (1 << PC4);

    while ((PINC & (1 << PC4)) != 0) {
    }
    motor(power, power);
    delay_ms(500);
  }

  //while any bumper is triggered
  while (((PINC & (1 << PC1)) == 0) | ((PIND & (1 << PD3)) == 0) | ((PIND &
(1 << PD4)) == 0) | ((PIND & (1 << PD5)) == 0) | (analog(5) > 500)) {
    //stop
    motor(0, 0);
    //turn on EVShield lights
    PORTC ^= (1 << PC5);
    //toggle beep
    beep(500, 250000);
```

```
    beep(0, 250000);
   }
 }
 disableMotor();
}
```

EVShield Program

```
#include <Wire.h>
#include <EVShield.h>
#include <EVs_NXTCam.h>
#include <USART.h>
#include <mechbotShield.h>
#include <avr/io.h>

EVShield evshield(0x34, 0x36);
EVs_NXTCam myCam; //make object myCam

// declare variables
int nblobs;
uint8_t color[8];
uint8_t left[8];
uint8_t top[8];
uint8_t bottom[8];
uint8_t right[8];
uint8_t i;
uint8_t LS = 0;
uint8_t RS = 0;


void setup()      {
 initUSART();
 evshield.init(SH_HardwareI2C);

 // reset motors
 evshield.bank_a.motorReset();
 evshield.bank_b.motorReset();

 myCam.init( &evshield, SH_BAS1 );
 // if there was previous run of this program, disable it.
 myCam.disableTracking();

 // setup myCam for Object mode and sortby size.
 myCam.selectObjectMode();

 myCam.sortSize();
 myCam.enableTracking();
 delay(1000);
```

```
 DDRB &= ~(1 << PB0);
 DDRB &= ~(1 << PB1);
 PORTB |= (1 << PB1);
 PORTB |= (1 << PB0);
}



void loop()    {
 myCam.issueCommand('J');
 // lock buffer
 delay(500);
 myCam.getBlobs(&nblobs, color, left, top, right, bottom);
 delay(500);
 myCam.issueCommand('K');

 while ((PINB & (1 << PB0)) == 1) { // PB0 is high == MechBot stopped

   myCam.issueCommand('J');
   // lock buffer
   delay(500);
   myCam.getBlobs(&nblobs, color, left, top, right, bottom);
   delay(500);
   myCam.issueCommand('K');


   if (LS == 0 && RS == 0) { // ********************* FIRST STOP
************************

     evshield.bank_a.motorRunDegrees(SH_Motor_1,
                                     SH_Direction_Forward,
                                     SH_Speed_Medium, 90,
SH_Completion_Wait_For,
                                     SH_Next_Action_BrakeHold);
     delay(2000);

      // If can is on the right
     if (color[i] == 2)  { // red can detected
       // Close claw and move can to front
       evshield.bank_a.motorRunDegrees(SH_Motor_2,
                                       SH_Direction_Forward,
                                       SH_Speed_Full, 100,
SH_Completion_Wait_For,
                                       SH_Next_Action_BrakeHold);
       delay(2000);
       evshield.bank_a.motorRunDegrees(SH_Motor_1,
                                       SH_Direction_Reverse,
                                       SH_Speed_Medium,
                                       90,
                                       SH_Completion_Wait_For,
                                       SH_Next_Action_BrakeHold);
       delay(2000);

       LS = 1;
       delay_ms(200);
```

```
      PORTB &= ~(1 << PB0);
      break;
    }
    else if ((color[i] == 3) | (color[i] != 2))  { // blue can detected
      // Close claw and move can to front

      evshield.bank_a.motorRunDegrees(SH_Motor_2,
                                      SH_Direction_Forward,
                                      SH_Speed_Full, 100,
SH_Completion_Wait_For,
                                      SH_Next_Action_BrakeHold);
      delay(2000);

      evshield.bank_a.motorRunDegrees(SH_Motor_1,
                                      SH_Direction_Reverse,
                                      SH_Speed_Medium,
                                      90,
                                      SH_Completion_Wait_For,
                                      SH_Next_Action_BrakeHold);
      delay(2000);

      RS = 1;
      delay_ms(200);
      PORTB &= ~(1 << PB0);
      break;
    }
    else { // ****************** CAN IS NOT ON THE RIGHT
********************
      delay(500);

      /********* Turn arm left **********/
      evshield.bank_a.motorRunDegrees(SH_Motor_1,
                                      SH_Direction_Reverse,
                                      SH_Speed_Medium, 180,
SH_Completion_Wait_For,
                                      SH_Next_Action_BrakeHold);
      delay(2000);

      myCam.issueCommand('J');
      // lock buffer
      delay(2000);
      myCam.getBlobs(&nblobs, color, left, top, right, bottom);
      delay(2000);
      myCam.issueCommand('K');

      // If can is on the left
      if (color[i] == 2)  { // red can detected
        // Close claw and move can to front
        evshield.bank_a.motorRunDegrees(SH_Motor_2,
                                        SH_Direction_Forward,
                                        SH_Speed_Full, 100,
SH_Completion_Wait_For,
                                        SH_Next_Action_BrakeHold);
        delay(2000);
        evshield.bank_a.motorRunDegrees(SH_Motor_1,
                                        SH_Direction_Forward,
                                        SH_Speed_Medium,
```

```
                                        90,
                                        SH_Completion_Wait_For,
                                        SH_Next_Action_BrakeHold);
          delay(2000);


          LS = 1;
          delay_ms(200);
          PORTB &= ~(1 << PB0);
          break;
        }
        else if ((color[i] == 3) | (color[i] != 2))  { // blue can detected
          // Close claw and move can to front
          evshield.bank_a.motorRunDegrees(SH_Motor_2,
                                        SH_Direction_Forward,
                                        SH_Speed_Full, 100,
SH_Completion_Wait_For,
                                        SH_Next_Action_BrakeHold);
          delay(2000);
          evshield.bank_a.motorRunDegrees(SH_Motor_1,
                                        SH_Direction_Forward,
                                        SH_Speed_Medium,
                                        90,
                                        SH_Completion_Wait_For,
                                        SH_Next_Action_BrakeHold);
          delay(2000);

          RS = 1;
          delay_ms(200);
          PORTB &= ~(1 << PB0);
          break;
        }


        delay(200);
        PORTB &= ~(1 << PB0);
      }
    }

    // ********************************** DROP OFF STATION
********************************** //
    else {
      if (LS = 1) {
        // Move arm left
        evshield.bank_a.motorRunDegrees(SH_Motor_1,
                                        SH_Direction_Reverse,
                                        SH_Speed_Full, 90,
SH_Completion_Wait_For,
                                        SH_Next_Action_BrakeHold);
        delay(2000);

        // Open claw to drop can
        evshield.bank_a.motorRunDegrees(SH_Motor_2,
                                        SH_Direction_Reverse,
                                        SH_Speed_Full, 5,
SH_Completion_Wait_For,
                                        SH_Next_Action_BrakeHold);
```

```
      delay(100);
      evshield.bank_a.motorRunDegrees(SH_Motor_2,
                                      SH_Direction_Reverse,
                                      SH_Speed_Full, 90,
SH_Completion_Wait_For,
                                      SH_Next_Action_BrakeHold);
      delay(1000);

      // Move arm right
      evshield.bank_a.motorRunDegrees(SH_Motor_1,
                                      SH_Direction_Forward,
                                      SH_Speed_Full, 100,
SH_Completion_Wait_For,
                                      SH_Next_Action_BrakeHold);
      delay(2000);

      LS = 0; // reset counter
      break;
    }
    else if (RS == 1) {
      // Move arm right
      evshield.bank_a.motorRunDegrees(SH_Motor_1,
                                      SH_Direction_Forward,
                                      SH_Speed_Full, 100,
SH_Completion_Wait_For,
                                      SH_Next_Action_BrakeHold);
      delay(2000);

      // Open claw to drop can
      evshield.bank_a.motorRunDegrees(SH_Motor_2,
                                      SH_Direction_Reverse,
                                      SH_Speed_Full, 5,
SH_Completion_Wait_For,
                                      SH_Next_Action_BrakeHold);
      delay(100);
      evshield.bank_a.motorRunDegrees(SH_Motor_2,
                                      SH_Direction_Reverse,
                                      SH_Speed_Full, 90,
SH_Completion_Wait_For,
                                      SH_Next_Action_BrakeHold);
      delay(1000);

      // Move arm left
      evshield.bank_a.motorRunDegrees(SH_Motor_1,
                                      SH_Direction_Reverse,
                                      SH_Speed_Full, 100,
SH_Completion_Wait_For,
                                      SH_Next_Action_BrakeHold);
      delay(2000);

      RS = 0; // reset counter
    }

    delay(500);
    PORTB &= ~(1 << PB0);

  }
```

```
 }
 else if ((PINB & (1 << PB1)) == 1) {
     //light up both LED's red
     evshield.ledSetRGB(255, 0, 0);
 }
 else {

   // Green flashing to indicate moving MechBot
   for (i = 0; i < 5; i++) {
     //light up both LED's
     evshield.ledSetRGB(0, 255, 0);
     delay(300);
     evshield.ledSetRGB(0, 0, 0);
   }
 }
}
```