Operation Manual

Team 11
Weeding Robot

Members:

Ian Nowak – ian10
Coen Purvis – cvp11
Amanda Richards – amr10e
Grant Richter – gmr10d
Jeremy Rybicki – jnr11b
Nathan Walden – new12

Faculty Advisor/s

Dr. Clark
Dr. Frank

Sponsor/s

Jeff Phipps

Instructors

Dr. Gupta

Date Submitted: April 3, 2015

Table of Contents

Table of Figures

Table of Tables

Acknowledgments

Thank you to the FAMU-FSU College of Engineering and Mechanical Engineering Department for helping to conceive this project. Thank you also to the machine shop, who has helped us fabricate the parts we needed. We should also thank Yoel Bugin for his help with water jetting many of our parts.

Finally, thanks to Dr. Clark, our advisor, and Mr. Jeff Phipps, our sponsor, because this project would not have been possible without them.

1. Introduction

The idea for this Senior Design project is to design and build a method for getting rid of weeds between the rows of crops on organic farms. Research tells us the idea of integrating robotic systems onto farms to reduce the labor and human dependency is not a new one. A lot of the existing technology will help guide us in the right directions for the purpose of our design.

Organic farms do not use traditional farming techniques such as herbicides and pesticides, so this robot will eliminate the need for a human to pull weeds from the farm plots. The robot will have to navigate between the rows of crops, remove weeds, keep itself charged and running 24/7, as well as follow other design constraints as outlined in this paper. Some of the challenges associated with these desired operations is the method of which the robot will be programmed to navigate through the plot. The team is composed of four mechanical engineers and two electrical engineers, and is sponsored by the mechanical engineering department. The project is sponsored by Jeff Phipps, of the Orchard Pond Organics farm, and is advised by Dr. Clark and Dr. Li.

2. Functional Analysis

The autonomous weeding robot is an all in one solution to reduce the amount of man power needed on an organic farm. The weeding robot uses a basket weeding design to uproot the shallow root systems of young weeds. This systems is meant to be deployed on a newly formed field where weeds have not become established yet. This robot utilizes the routine maintenance approach to keep the field weed free like the day it was first plowed. The user needs to place markers within the field to add an extra layer of protection for crops. The robot uses a vision system in unison with other ultrasonic ranging modules to accurately navigate the field. The vision system uses the markers placed within the field make sure that the ranging module is producing accurate data. The vision system can navigate without using the ultrasonic sensors and vice versa. The reason for using multiple sensors is to increase the accuracy and reliability of the robot. A visual example of this analysis can be found in Appendix A.

3. Project/Product Specification

Without doubt, the most important part of the weeding robot is the weeding mechanism, which is the basket attached at the side of the robot. The spokes of the basket are 6 inches long, and the overall diameter of the basket is 6.5 inches. These are the most important dimensions because they determine the way that the mechanism will weed the plant bed. Firstly, because the spoke is 6 inches long, it can weed that far into the bed, and will not be able to reach the plants in the middle. Secondly, since the outside diameter of the basket is 6.5 inches, the spokes will be able to penetrate at most 1 inch into the soil. This is because the bed varies from 3.5 – 4 inches and the basket is mounted so that the bottom spoke is 4.5 inches above the ground. This ensures that the robot will meet the sponsor’s desire that soil below 1 inch will not be disturbed.

The motor being used to turn the weeding mechanism is Part # 402626 from Maxon motors. Without the gearhead, this has a stall torque of 8920 mNm. With the gearhead (ratio of 43:1), the motor has a stall torque of 383560 mNm and a stall current of 232 A. However, the nominal torque/current is 405 mNm/10.8A.

The microcontroller being used is the BeagleBone Black, the specifications for which can be seen in Fig. #. The most significant specification of the BeagleBone Black is the fact that is has 1 GHZ of processing speed which allows for real time image processing. The motor capes used to control the motors are the DMCC Mk. 7. The specifications can be found in Appendix C, Table 1. This part has dual channel motor capes with the capabilities to drive a wide range of motors.

4. Product Assembly

1.1 3D Model

The 3D model and the associated exploded view can be found in Appendix C.

1.2 Crucial Component Assembly

1.2.1 Tools needed

- Hex Key
- Phillips head screw driver
- Bearing press
- Wrench
- Welding tools

1.2.2 Assembling the frame

The Dagu Wild Thumper robot frame is the first aspect of the weeding robot that must be assembled. The frame of the robot is mostly assembled out of the box, and all that needs to be assembled is coupling the wheels to the motor. This step only requires the hex key.

Next, the aluminum plates need to be attached to the top of the robot. These will support the weeding mechanism and the motor. Since the top plate comes attached to the robot, it will need to be removed. From the bottom of this plate, align the two aluminum plates with the appropriate holes on the top plate. Insert 2 Phillips head screws into the appropriate holes on one aluminum plate, and then on the other. This top plate can be reattached to the robot.

Using a bearing press, the bearings need to be placed in the appropriate holes. There are two bearings necessary, one on each aluminum plate.

1.2.3 Inserting electronics/Hooking up motors

The motor that will power the basket goes through the larger of the holes on the aluminum plates. An additional circular plate is used. Using Phillips head screws, fix the circular plate to the face of the motor. Following this, again using Phillips head screws, attach the circular plate to the aluminum plate. The fitting for the motor shaft (which includes the sprocket) can be slipped over the shaft. This can be fixed at the end of the motor shaft by placing a washer at the end and fixing it with a screw into the motor shaft.

1.2.4Assembling the basket

To assemble the basket part of the weeding mechanism, the metal hubs, metal spokes and carriage bolts are needed. Align the square holes on the spoke with the square holes in the hub and couple with the carriage bolt/nut. Tighten with a wrench. Do the same with the other hub.

Using welding tools, the basket should be coupled onto the end of the shaft. The shaft should then be press fit into the bearings. Following this, the sprocket should be welded (?) inches from the opposite end of the shaft. The chain can now be fit over the two sprockets, and the motor should now be able to spin the basket.

5. Operation Instructions

- Field set up

The farmer should first take the orange markers provided and place them in the middle of each bed spaced every 10 feet along the row

- Robot Orientation

Before the robot is turned on it has to be lined up in the first row. To do this the robot should be tuned so that its webcam is facing the down the row. Next the robot should be centered in the furrow. Finally the robot is placed so that the weeding mechanism is touching the first bed on its right.

- Powering on

Press the red button and watch for the led to signal that the robot is on.

- Monitoring

There should be someone watching the robot to ensure that it doesn't go off track and that it makes it to the end of the row.

- End of row

Once it is at the end of the row the robot should turn and go down the next row if it does not the user must pick it up and repeat the operation instructions starting at Robot Orientation.

- End of plot

Now that the robot has finished weeding the entire plot the user should power off the robot and charge the batteries for later use.

6. Troubleshooting

1.3 Mechanical Component Problems

What are the potential problems of your project?

- Wheels
  - Come loose
- Chassis
  - Screw could come loose
- Basket
  - Spokes could deform
  - Could accumulate debris
  - Not rotating properly
- Chain
  - Things can get stuck
  - Can slip off gears
  - Can rust or break

How to solve those problems when those problems occurs?

- Wheels
  - Make sure wheel is securely fastened to the axle
- Chassis
  - Replace with extra screws
- Basket
  - Replace spokes
  - Turn off power to motor, use a brush to clean debris from spokes
  - Check to see if
    - Power is on
    - Robot is moving
      - If not, charge batteries
      - If moving, check to see if chain is still attached, shaft is still aligned, and chain is free of debris
- Chain
  - Turn power off to motor, check chain. Remove any debris
  - Rotate sprocket with hand and align gear as it rotates
  - Buy a new chain at local hardware store with design specification

1.4 Electrical Component Problems

What are the potential problems of your project?

- Navigation
  - Robot traverses over beds
  - Robot does not move
- BeagleBone
  - No lights coming from computer
- Motors
  - Two motors do not spin, signifying the battery is low or dead

How to solve those problems when those problems occurs?

- Navigation
  - Check that robot is powered on. Make sure ultrasonic sensor is clear of debris. If necessary, clean with dry cloth.
  - Check that camera is clear from obstruction and ensure the orange balls are not covered. Also make sure the webcam is plugged into the USB port.
- BeagleBone
  - Charge batteries
- Motors
  - Charge batteries

7. Regular Maintenance

For optimal operation, it would be best to complete a number of actions approximately once per week. An overview of the key components that should be evaluated before operation of the autonomous weeding robot are broken down into three categories.

1.5 Maintenance of Categories

A. Chassis
   - Suspension check – Frame suspension should always be locked, make sure there is no horizontal or vertical movement.
   - Aluminum support – Make sure support is still securely attached to the frame.

B. Basket Weeding Mechanism
   - Spokes – Check for any bending or shearing that may have occurred during previous operation.
   - Coupling – Check to make sure the bolts are all tightened to ensure the spokes are fastened completely.

C. Electrical Components
   - Battery – Routine battery charging weekly.
   - Webcam – Check to make sure there is no damage to the face and ensure there is no debris.
   - Motors – Make sure they are still rotating with the correct speed and efficiency.

1.6 Key Component Replacement

The rugged terrain and environment of which the robot will be operating in makes it evident that parts will often need replacement as it is likely the parts will become worm from the weather and other outside conditions. The key components that might need replacement over others would be the basket weeding mechanism. This part will experience the most wear and the most degradation from the soil. We have made this part very modular by making the connections simple and the material easily attainable. The consumer can have the option of making the spokes larger smaller, depending on their desired application.

1.7 Spare Parts Inventory

The Table in Appendix C, Table 2, shows spare parts that where needed to ensure operation is minimally disrupted.

8. Conclusion

Ultimately, for this senior design project the goal of the weeding robot team was to create an autonomous robot that could effectively remove weeds from a plot. The method chosen to remove weeds from the plot was the basket weeding method. The basket weeding method uses spokes that dig into the ground and sweep the weeds out from their root system. This basket is driven by a motor that is powerful enough to overcome the torque needed to drive the mechanism into the dirt. A Beaglebone Black is used to control the robot as well as the motor driving the weeding mechanism. The robot navigates through the plot by using a vision system in parallel with an ultrasonic sensor. The vision system uses a webcam to identify markers on the sides of the row, and then compares it with the data from the ultrasonic sensor in order to autonomously drive the robot straight through the furrow. By using these two systems together they are able to correct any errors or bad data from one of the sensors alone. The mechanical and electrical designs work together to affect 100% of the weeds that the robot encounters.

References

References can be written in single space with extra space between references as in the format below. There are many different ways to arrange the information and punctuation in a reference listing. The most important thing is to make sure all references are complete and that the format of your references is consistent throughout. See additional suggestions and possible formatting options online.

Example:

[1] N. Gupta. Dynamic modeling and motion planning for robotic skid-steered vehicles, Ph.D. dissertation, Florida State University, Tallahassee, FL, June 2014.

[2] C. Ordonez, N. Gupta, W. Yu, O. Chuy, and E. Collins. Modeling of skid-steered wheeled robotic vehicles on sloped terrains. In Proceedings of the ASME Dynamic Systems and Control Conference, pages 91–99, 2012.

Appendix A

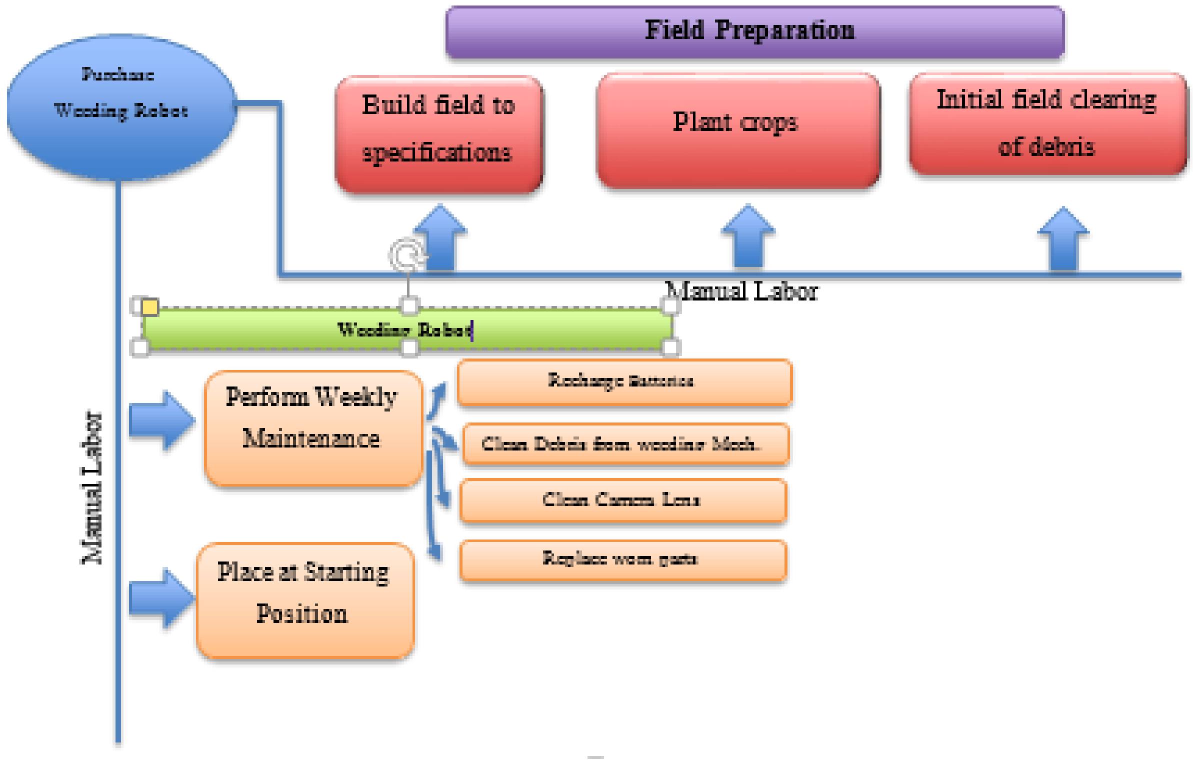

Field Preparation
Purchase Weeding Robot
Build field to specifications
Plant crops
Initial field clearing of debris
Manual Labor
Weeding Robot
Manual Labor
Perform Weekly Maintenance
Recharge Batteries
Clean Debris from weeding Mech.
Clean Camera Lens
Replace worn parts
Place at Starting Position

Figure 1. User Functional Structure (Macro)

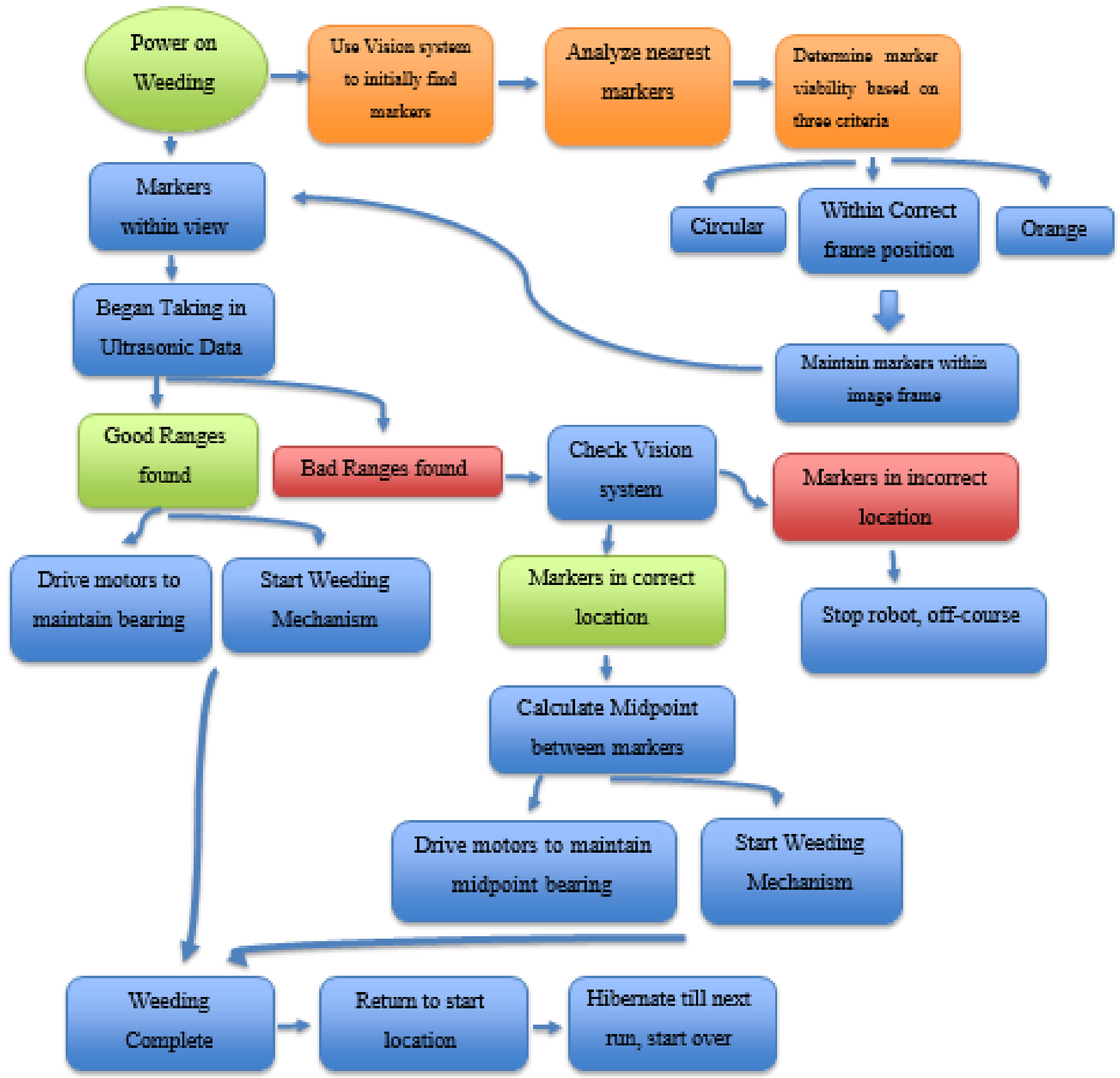
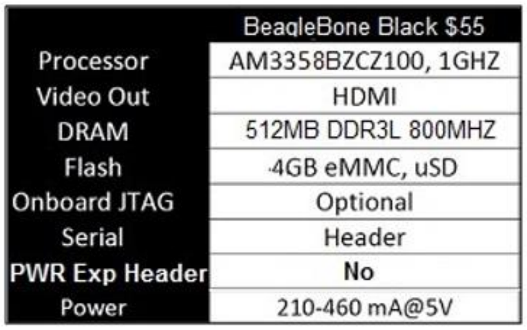

Power on Weeding
Use Vision system to initially find markers
Analyze nearest markers
Determine marker viability based on three criteria
Circular
Within Correct frame position
Orange
Maintain markers within image frame
Markers within view
Began Taking in Ultrasonic Data
Good Ranges found
Bad Ranges found
Check Vision system
Markers in incorrect location
Stop robot, off-course
Markers in correct location
Drive motors to maintain bearing
Start Weeding Mechanism
Calculate Midpoint between markers
Drive motors to maintain midpoint bearing
Start Weeding Mechanism
Weeding Complete
Return to start location
Hibernate till next run, start over

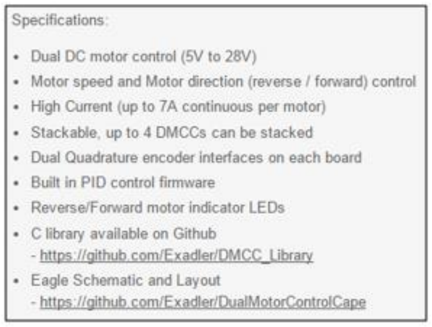
Figure 2. Weeding Robot Functional Structure (Micro)

Appendix B

Table 1. Specifications for Microcontroller and Motor Capes



	BeagleBone Black $55
Processor	AM3358BZCZ100, 1GHZ
Video Out	HDMI
DRAM	512MB DDR3L 800MHZ
Flash	4GB eMMC, uSD
Onboard JTAG	Optional
Serial	Header
PWR Exp Header	No
Power	210-460 mA@5V



Specifications:

- Dual DC motor control (5V to 28V)
- Motor speed and Motor direction (reverse / forward) control
- High Current (up to 7A continuous per motor)
- Stackable, up to 4 DMCCs can be stacked
- Dual Quadrature encoder interfaces on each board
- Built in PID control firmware
- Reverse/Forward motor indicator LEDs
- C library available on Github
  - https://github.com/Exadler/DMCC_Library
- Eagle Schematic and Layout
  - https://github.com/Exadler/DualMotorControlCape

Table 2. Spare Parts Inventory

Component	Item	Unit Cost $	Quantity	Total Cost $	Supplier
Electrical	BeagleBone Black	55.00	1	55.00	Adafruit
	Motors	Varies	1-3	Varies	Pololu
	Webcam	40.00	1	40.00	Amazon
Mechanical	Spokes	Varies	Varies	Varies	Home Depot
	Wheels	15.00	1	15.00	Pololu
	Screws for Chassis	Varies	1-8	Varies	Pololu
	Spare chain	20.00	1	20.00	Local Hardware Store

Appendix C

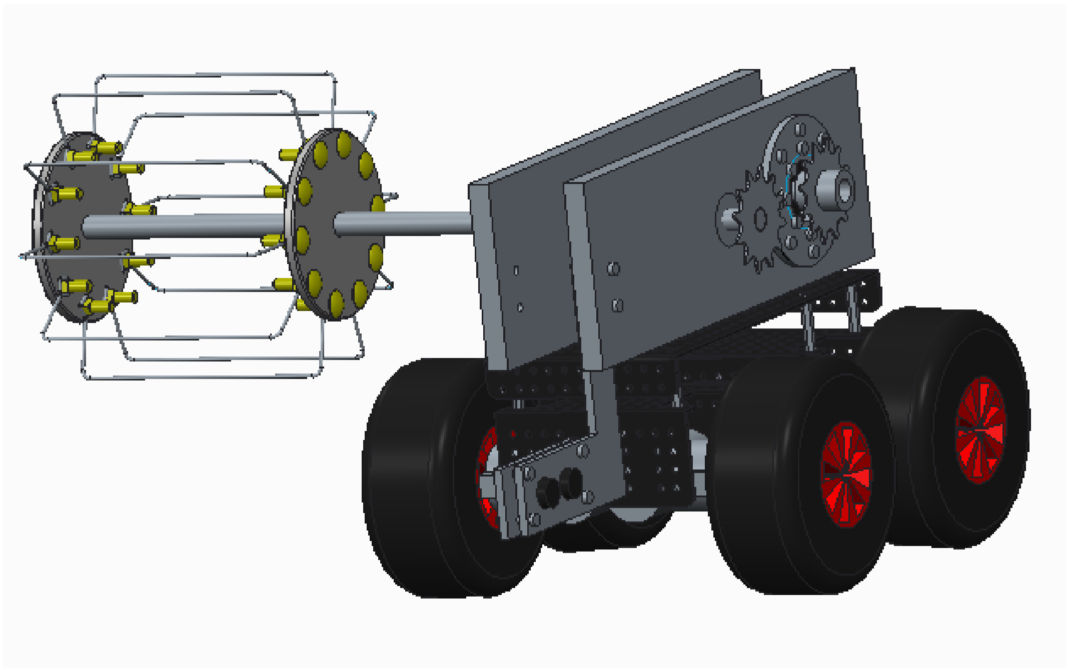


Figure 3. 3D Model

Appendix

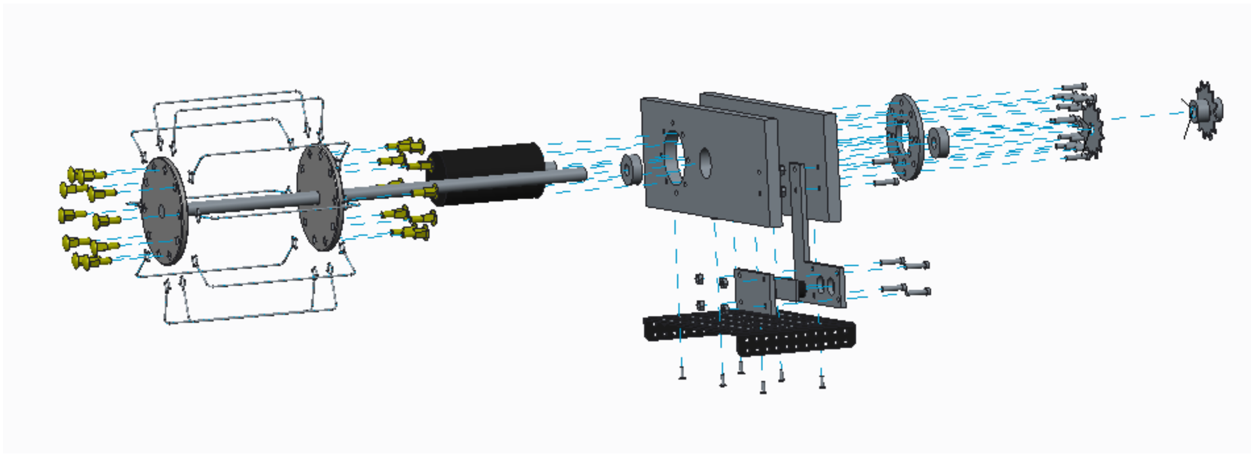


Figure 4. Full Assembly

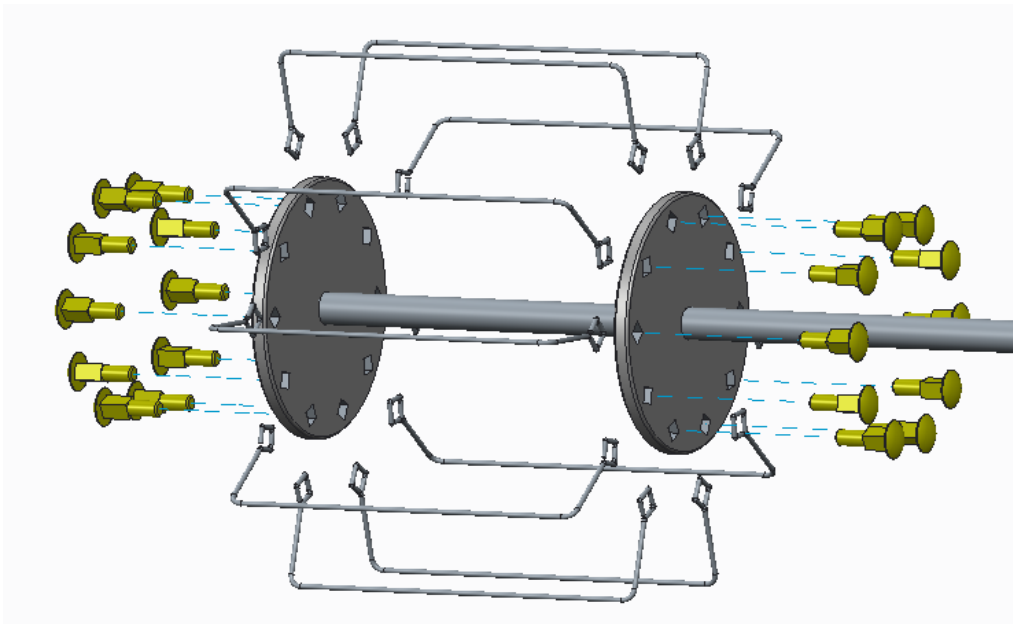


Figure 5. Basket Assembly

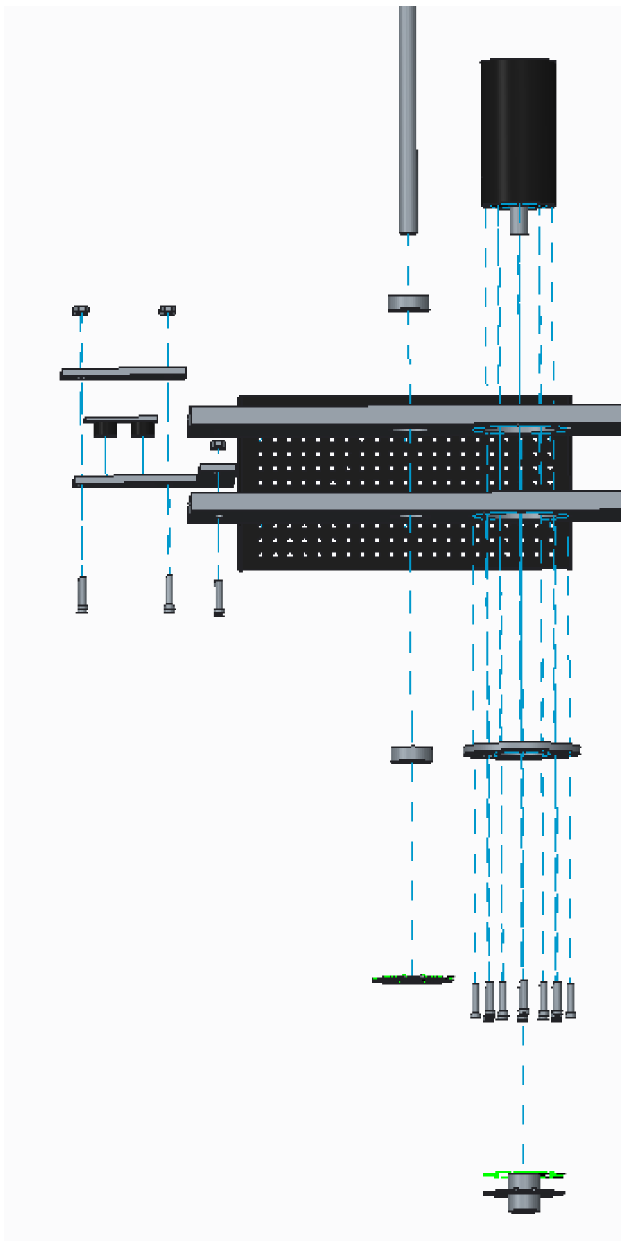
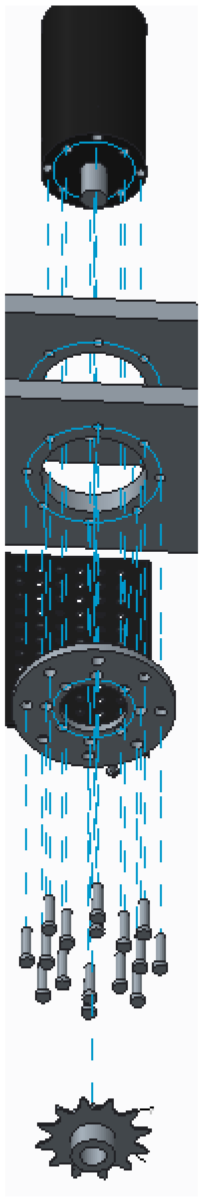



Figure 6. Top and Motor Mounting View

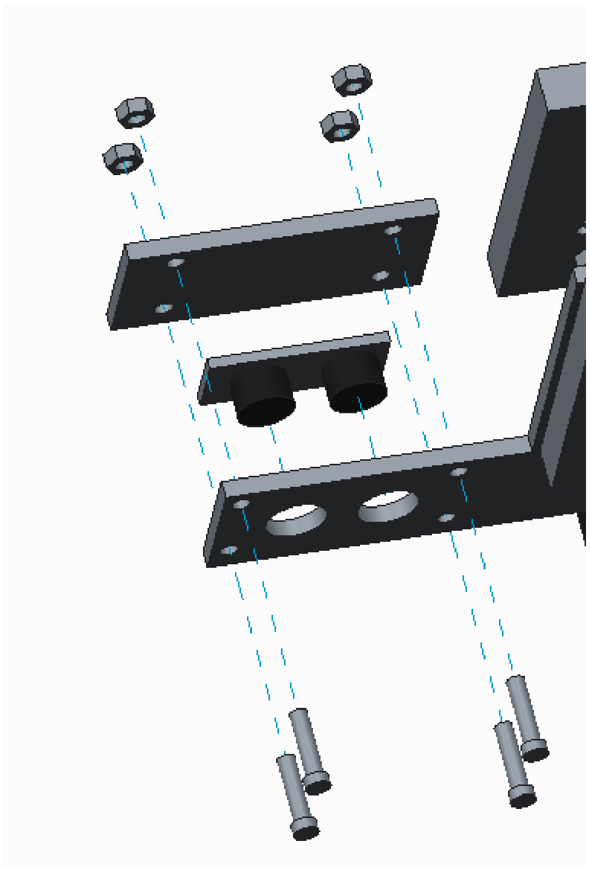


Figure 7. L-Bracket Assembly

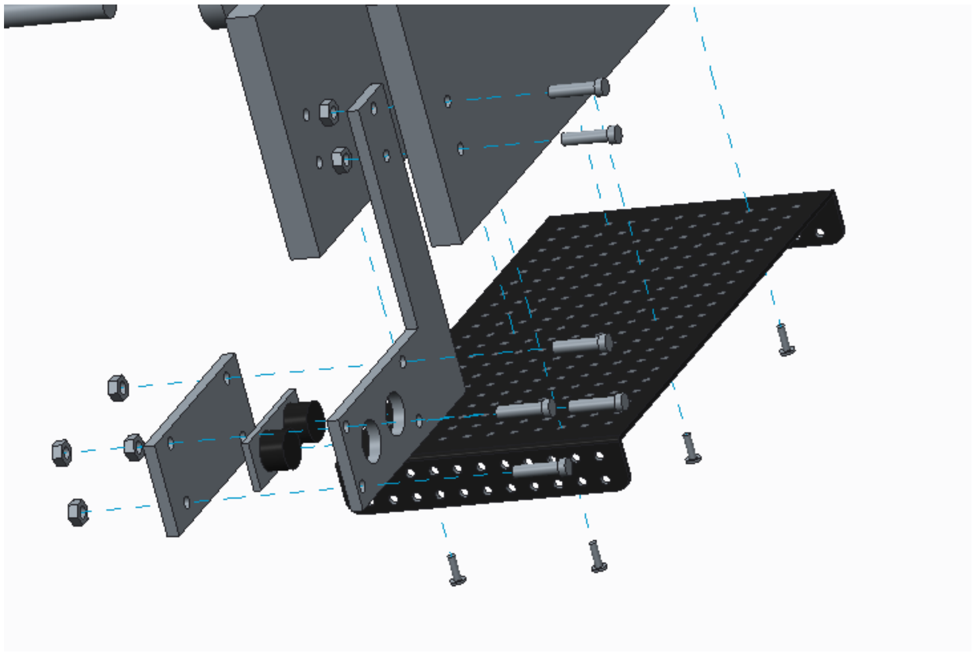


Figure 8. Bottom Plate Assembly

Appendix D

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <cmath>
#include "DMCC.h"
#include <prussdrv.h>
#include <pruss_intc_mapping.h>
#include "GPIO/GPIOManager.h"
#include "GPIO/GPIOConst.h"

//standard and openCV namespaces
using namespace std;
using namespace cv;
void Mdrive(int,int,int);
int ultradrive(float);
float ultradistance();
//declare some global variables which will be used
double width = 1280;    //camera sizes
double height = 720;
int thresh = 100;       //threshold values for colors
int max_thresh = 255;
int speed = 4400;       //default robot speed
int noinput = 0;        //checks for bad data
int badsonic = 0;
int turnratio = 0.65;   //defulat turn speed
int ultraset = 0;       //check if ultrasonic sensor is set

int main(){
	//set camera and size
	VideoCapture capture(0);
	capture.set(CV_CAP_PROP_FRAME_WIDTH, width);
	capture.set(CV_CAP_PROP_FRAME_HEIGHT, height);
	double width = capture.get(CV_CAP_PROP_FRAME_WIDTH);
	double height = capture.get(CV_CAP_PROP_FRAME_WIDTH);

	Mat imgInput, imgResize, imgHSV,  imgNew;
	Mat canny_output, threshold_output;
	vector<vector<Point> > contours;
	vector<Vec4i> hierarchy;
	sleep(5);
	while(1){
		//check if camera is detected
		if(!capture.isOpened()){
			cout << "No Camera" << endl;
		}

		//capture image here
		capture >> imgInput;

		//resize image
```

```
            Size size(640, 360);
            resize(imgInput,imgResize,size);

            //convert to HSV color space
            cvtColor(imgResize, imgHSV, CV_BGR2HSV);
            inRange(imgHSV, Scalar(5, 130, 90), Scalar(17, 255, 255), imgNew);

            // Detect edges using canny
            Canny(imgNew, canny_output, thresh, thresh*2, 3 );
            // Find contours
            findContours(  canny_output,  contours,  hierarchy,  CV_RETR_TREE,
CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );

            //Creaste contours for polygons + get bounding rects and circles
            vector<vector<Point> > contours_poly(contours.size());
            vector<Rect> boundRect(contours.size());
            vector<Moments> mu(contours.size());
            vector<Moments> mu2(contours.size());
            vector<Point2f>mc(contours.size());
            vector<Point2f>mc2(contours.size());
            vector<Point2f>center(contours.size());
            vector<float>radius(contours.size());

            //generate circle around center of object
            for(int i = 0; i < contours.size(); i++){
                  minEnclosingCircle(  (Mat)contours[i],  center[i],  radius[i]
);
            }

            int temp = 0;
            //j is the largest object, k is the second largest
            int j = 0;
            int k = 0;
            //find the largest object
            for(int x = 0; x < contours.size(); x++){
                  if(contourArea(contours[x]) > temp){
                        temp = contourArea(contours[x]);
                        j = x ; //set the new largest object
                        mu[j] = moments( contours[j], false );
                        mc[j]     =     Point2f(     mu[j].m10/mu[j].m00     ,
mu[j].m01/mu[j].m00 );
                  }
            }
            temp = 0;
            //find the second largest object
            for(int y = 0; y < contours.size(); y++){
                  if(contourArea(contours[y]) > temp){
                        if(y != j){  //ignore the largest contour
                              mu2[y] = moments( contours[y], false );
                              mc2[y]   =   Point2f(  mu2[y].m10/mu2[y].m00   ,
mu2[y].m01/mu2[y].m00 );
                              //check to see that countours aren't the same
                              if(abs(mc[j].x - mc2[y].x) > 60){
                                    temp = contourArea(contours[y]);
                                    k = y ; //set the new largest object
                              }
                        }
                }
        }
        //check if we found two objects
        if((j != 0) && (k != 0)){
                cout << "found objects" << endl;
                //make sure objects found aren't just random blobs
                if((contourArea(contours[j])          >          1)          &&
(contourArea(contours[k]) > 1)){

                        //check if the balls are on the side of view
                        if(((mc[j].x > 427) && (mc2[k].x < 213)) || ((mc2[k].x
> 427) && (mc[j].x < 213))){
                                //loop ultrasonic code unless weird data
                                while(true){

                                        //call function to get ultrasonic sensor
                                        float ultradis = ultradistance();

                                        //call motor drive controller, if bad data
exit while loop
                                        if(ultradrive(ultradis) == 1){
                                                break;
                                        }
                                }
                        }
                        //reset bad data counter
                        noinput = 0;
                }
        }
    }

}

//setup the ultrasonic sensor and call the asm program
//to return the data timing value in order to be converted
//into a distance usable by the robot
float ultradistance(){
    /* Get measurements */
    printf(">> Initializing PRU\n");
    tpruss_intc_initdata pruss_intc_initdata = PRUSS_INTC_INITDATA;
    prussdrv_init();
    /* Open PRU Interrupt */
    if (prussdrv_open (PRU_EVTOUT_0)) {
        // Handle failure
        fprintf(stderr, ">> PRU open failed\n");
        return 1;
    }
    /* Get the interrupt initialized */
    prussdrv_pruintc_init(&pruss_intc_initdata);
    /* Get pointers to PRU local memory */
    void *pruDataMem;
    prussdrv_map_prumem(PRUSS0_PRU0_DATARAM, &pruDataMem);
    unsigned int *pruData = (unsigned int *) pruDataMem;
    /* Execute code on PRU */
    printf(">> Executing HCSR-04 code\n");
```

```
	prussdrv_exec_program(0, "hcsr04.bin");

	// Wait for the PRU interrupt
	prussdrv_pru_wait_event (PRU_EVTOUT_0);
	prussdrv_pru_clear_event(PRU_EVTOUT_0, PRU0_ARM_INTERRUPT);

	// Print the distance received from the sonar
	// At 20 degrees in dry air the speed of sound is 342.2 cm/sec
	// so it takes 29.12 us to make 1 cm, i.e. 58.44 us for a roundtrip of 1
cm
	float ultradis = ((float) pruData[0] / 58.44);
	return ultradis;
}


//controller function to drive the robot based on the
//ultrasonic distance value from the sides of the beds
int ultradrive(float disvalue){
	if(badsonic > 5){
		Mdrive(0, 0, speed);
		return 1;
	}
	else if(disvalue < 25){
		turnratio = (0.65 - (((19-disvalue)/19)*.2));
		Mdrive(1, 1, speed);
		badsonic = 0;
	}
	else if(disvalue > 50){
		badsonic++;
	}
	else if(disvalue > 30){
		turnratio = (0.65 - (((disvalue - 24.5)/24.5)*.2));
		Mdrive(1, 2, speed);
		badsonic = 0;
	}
	else{
		turnratio = 0.65;
		Mdrive(1,0,speed);
		badsonic = 0;
	}
	return 0;
}


//Mdrive is the function used to control the motor capes which
//are placed on top of the beaglebone. It is possible to set
//the motor power, direction, and cape with this function call
void Mdrive(int direction, int turn,  int speed){

	int session = DMCCstart(0x00);

	switch(direction){
		case 0:
			setMotorPower(session, 1 ,0000);
			setMotorPower(session, 2 ,0000);
			break;
```

```
            case 1:////forward
                  setMotorPower(session, 1 ,0000);
                  setMotorPower(session, 2 ,0000);
                  if(turn == 0){////straight
                        setMotorPower(session, 1 ,speed);
                        setMotorPower(session, 2 ,(-1)*speed);
                  }
                  else if(turn == 1){/////right
                                setMotorPower(session, 1 ,speed*0.65);
                                setMotorPower(session, 2 ,(-1)*speed);
                        }
                  else if(turn == 2){/////left
                                setMotorPower(session, 1 ,speed);
                                setMotorPower(session, 2 ,(-1*0.65)*speed);
                        }

                  break;
            case 2:////backward
                  setMotorPower(session, 1 ,0000);
                  setMotorPower(session, 2 ,0000);
                  if(turn == 0){////straight
                        setMotorPower(session, 1 ,(-1)*speed);
                        setMotorPower(session, 2 ,speed);
                  }
                  else if(turn == 1){/////right
                                setMotorPower(session, 1 ,speed*(-1*0.65));
                                setMotorPower(session, 2 ,speed);
                        }
                        else if(turn == 2){/////left
                                setMotorPower(session, 1 ,-1*speed);
                                setMotorPower(session, 2 ,(0.65)*speed);
                        }
                  break;
            case 3: // turn left
                  setMotorPower(session, 1 ,0000);
                  setMotorPower(session, 2 ,0000);
                  if(turn == 0){////straight
                        setMotorPower(session, 1 ,speed);
                        setMotorPower(session, 2 ,speed);
                  }
                  break;
            default:
                  break;
      }

      DMCCend(session);
}
```