Autonomous Recycling Robot Final Report

By:

Antaeus Kleinert-Strand,

Mochamad Adrian Prananda,

Tzu-Wei Chuang

Table of Contents

Page #

ABSTRACT

This is the final report for the autonomous recycling robot project. This robot is able to recognize a banana or a bottle, trap it, and bring it to a designated location. For our design, the environment will be predefined, meaning that the robot will have already learned its environment, including walls, barriers, and other obstacles. It will be able to find these predetermined locations using a compass. Image processing will be done on the Raspberry Pi, while system controls will be handled by the MSP432. Included in this report are the basic descriptions of how this project will be implemented, comprising of discussion of the lab, test plan, analysis of the results, bill of materials and a schedule for production.

INTRODUCTION

The purpose of this project is to build an autonomous robot using a TI MSP432 board and the Raspberry Pi that is able to trap certain objects and move them to a desired location. Input data from digital distance sensors around the robot will help it navigate its environment, and a camera takes pictures for object recognition. The robot's current actions are displayed on an OLED attached in the back, along with a compass to help with navigation and location. Servo motors are used to control the motion of the robot and the trapping mechanism. The two boards communicate through a UART connection, with the MSP432 asking for object recognition compass data from the Raspberry Pi. This device is designed to work within a defined environment for specific object recognition.

DISCUSSION OF THE LAB

Requirement Specifications

System Description

This specification describes the requirements for a prototype object-recognition sorting robot. This robot is able to recognize objects based on imported library data, and relocate them into predefined areas. Visual recognition, distance sensors for object avoidance, and a mode of controlled motion are required for this robot. It should be able to move autonomously while searching for the designated objects. The robot will locate and trap the object, moving it to a collection spot. This should be done using some sort of tracking/navigation system on the robot. To save power camera use should be limited to when an object is directly in front of the robot, and not a wall. This device should be low power for extended mobility.

Specification of External Environment

This device should be useable in many diverse environments. Object recognition must be very exact, with little error in identification. The most typical environment will be indoors, with multiple objects to navigate around. It can be used in either commercial or consumer environments. With modifications to a base model, it should be able to work in rougher terrain. Power should be handled through a rechargeable battery pack, in order to provide fast battery swapping and relatively low down times.

System Input and Output Specification

System Inputs

The system will be able to measure and read the following signals.

Camera Module:

- Supplied by a voltage ranging from 2.7 V to 3.3 V

- Supports a .jpeg format with image size of 320 x 240 pixels
- Mounted on the front of the robot angled down
- Capture the image of the to-be-sorted objects

Distance Sensor:

- Supply voltage of 5 V
- Sampled at 200 Hz
- Range 5 cm - 10 cm from the object
- Stable signal with quick response time
- Navigation: one mounted on the left and right sides, one mounted on the front right and left corners, and one mounted front center
- Object Detection: two sensors mounted below the platform - front center

Mode Select (Manual, Autonomous):

- Controlled through bluetooth command
- The user manually selects the functionality for either autonomous mode or user-control mode
- Manual mode - this configuration stops operation until prompted by the user reducing power consumption. The control is done through a GUI on the user's computer
- Autonomous mode - the robot will drive around itself, collect the object, and put it in the predefined location

Bluetooth Module:

- Supplied by a voltage ranging from 3.3 V to 6 V
- Generates signal with a frequency of 2.4 GHz
- Can communicate with other Bluetooth modules up to 18 m away
- Allows communication to the user's device that has Bluetooth capability (for example, a smartphone) so that the user can control the robot manually from the device
- Error Rate: the bit error rate is 0.0007%

Power ON/OFF:

- Switch controlling system power
- Powered by a rechargeable battery pack

Reset:

- A reset button on the robot sets it to manual mode (default)

System Outputs

The system shall measure and display the following signals

Object Interaction:

- Control a trapping mechanism - this is controlled by the object recognition capability of the robot, making it only trap and hold the designated object

Object Sorting:

- Be able to move the object to a designated location based on what it is
- Pre-define objects to collect. These objects are then sorted into separate collection areas
- Distinguish objects by looking at specific identifiers, these range from color, shape, size, and patterns on the object.

Robot Motion:

- Mechanism that allows the robot to move around on level, smooth terrain
- Able to focus in on a object in order to collect it
- Releasing the object once in the correct location

User Interface

The user will be able to select the mode the robot is in using a switch located on the robot. They will also be able to control the robot in manual mode with the use of a bluetooth controller. The user will decide which objects to pick up and where they should be returned to.

Use Cases

The use cases for the object collector system consists of two diagrams: the manual mode and the autonomous mode. Once the user selects autonomous mode, the user cannot interact with the robot, except to change modes. Only after being switched to manual mode will the user gain control over the robot's actions. The rest of the actions are done entirely by the robot. In manual mode the user has full control over all of the robot's actions. The user can control the movement of the robot such as turning left, turning right, going straight, going backward, lowering and lifting the trapping mechanism. The following diagrams show the connection between the user and the robot in autonomous mode and manual mode.

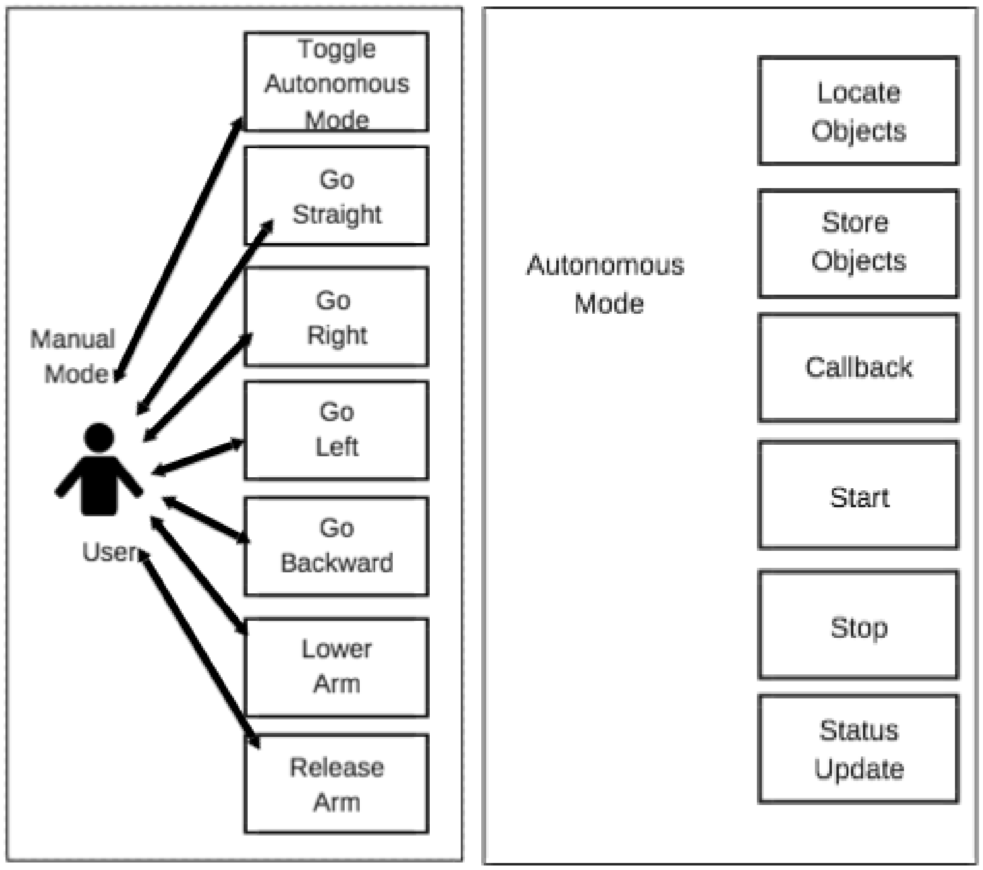


Figure 1: The use cases diagram for the autonomous object recognition device

Autonomous Mode

Once started the robot will move on its own until the battery runs out, or prompted to switch over to manual mode. No interaction is required from the user.

Manual Mode

These commands ignore all sensor input and will behave according to the input received from the user.

Toggle Manual Mode - Toggles the robot between manual and autonomous modes.

Go Straight - The robot will move in a straight line.

Go Right - The robot will turn in place to the right.

Go Left - The robot will turn in place to the left.

Go Backwards - The robot will move backwards in a straight line

Lower Arm - The user can lower the arm freely whether the robot currently have an object or not.

Lift Arm -The user can lift the scoop freely whether the robot currently have an object or not.

Operating Specifications

The device will operate in a standard commercial/consumer environment.

- Temperature range: 12 - 50 ℃
- Humidity up to 90% RH non-condensing
- Battery Power: 5V 10000mAh for a minimum of 8 hours fully-charged battery
- Bluetooth signal: 2.4 GHz
- Clear places with minimal impact from fog / smoke for camera vision

The system time base shall meet the following specifications:

- Temperature stability: 0-50 ℃< 6 x 10^-6
- Aging Rate:
- 90 day < 3 x 10^-8
- 6 month < 6 x 10^-7
- 1 year < 25 x 10^-6

Reliability and Safety Specification

- Safety: UL-3111-1, IEC-1010, CSA 1010.1
- EMC: CISPR-11, IEC 801-2, -3, -4, EN50082-1
- Bluetooth safety requirement: CENELEC EN 50385:2002
- Battery requirements: IEC 62133-2
- Mean Time Between Failures: minimum of 100,000 hours
- Max object weight: 1lb

System block diagram

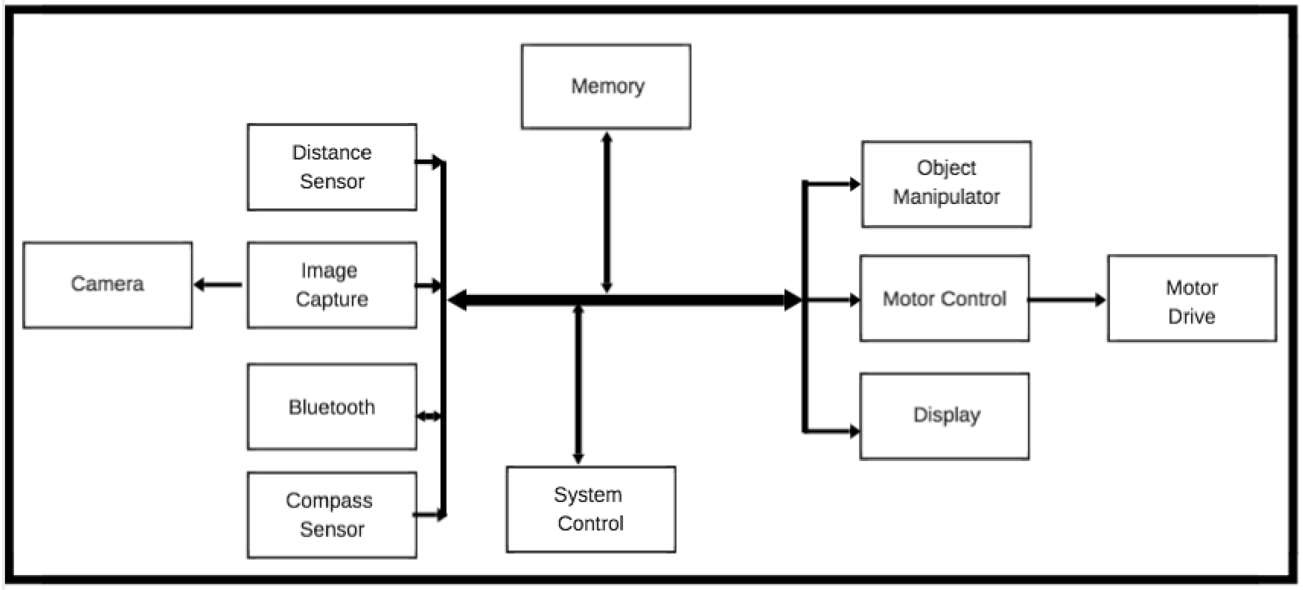


Figure 2: The system block diagram for the autonomous recycling robot

Design Specification

System Description

This specification describes the basic designs for a prototype object-recognition sorting robot. This robot is able to recognize objects based on machine learning from precompiled libraries, and sort them into predefined locations. Using a camera for visual recognition, digital distance sensors for object avoidance, and continuous rotation servos for wheels, the robot will be able to move autonomously while searching for the designated objects. The robot will locate and trap the object, moving it to the correct collection spot, e.g. the NW corner of the environment; this is done using a compass located on the robot. The digital sensors on the robot are used to prompt object recognition to start and to avoid walls and other obstacles within the testing environment. This device is low power for extended mobility.

Specification of External Environment

This device is meant to be used in many environments, however our prototype can only work on flat terrain with very simple surroundings. This allows for the object recognition to work more accurately and to allow the robot the greatest degree of freedom. The most typical environment will be indoors, with multiple objects to navigate around. It can be used in either commercial or consumer environments. With a more sturdy body, wheels designed for uneven terrain, and waterproofing, the EnviRobo can be used outside. Charging is done by recharging the detachable battery through a usb adapter charging block. The wheels battery supply is run off of 4 AA batteries, these can be replaced as needed.

System Input and Output Specification

System Inputs

The system will be able to measure and read the following signals.

Raspberry Pi Camera Module:

- Supplied by a voltage ranging from 2.7 V to 3.3 V
- Supports a .jpeg format with image size of 320 x 240 pixels
- Mounted on the front of the robot angled down
- Capture the image of the to-be-sorted objects

Distance Sensor:

- Supplied by a voltage of 5 V
- Sampled at 200 Hz
- Range 5 cm - 10 cm from the object
- 10 µF capacitor connected between VCC and ground, to add stability to the input signal - connected through breakout breadboard
- Navigation: one mounted on the left and right sides, one mounted on the front right and left corners, and one mounted front center
- Object Detection: two sensors mounted below the platform - front center
- Initiate object recognition software and navigate around the environment

Mode Select (Manual, Autonomous):

- Controlled through bluetooth command
- The user manually selects the functionality for either autonomous mode or user-control mode
- Manual mode - this configuration stops operation until prompted by the user reducing power consumption. The control is done through a GUI on the user's computer
- Autonomous mode - the robot will drive around itself, collect the object, and put it in the predefined location

Bluetooth Module:

- Supplied by a voltage ranging from 3.3 V to 6 V
- Generates signal with a frequency of 2.4 GHz
- Can communicate with other Bluetooth modules up to 18 m away
- Attached to a breakout breadboard
- Allows communication to the user's device that has Bluetooth capability (for example, a smartphone) so that the user can control the robot manually from the device
- Protocol: UART
- Error Rate: the bit error rate is 0.0007%

Power ON/OFF

- MSP432 and Raspberry Pi are powered by a rechargeable battery pack
- Continuous rotation servos are powered by 4 * AAA batteries

Reset:

- A reset button on the MSP432 sets it to manual mode (default)
- SSH used to reset the Raspberry Pi functionality

System Outputs

The system shall measure and display the following signals

Object Interaction:

- Use two servos to rotate a trapping mechanism - this is controlled by the object recognition capability of the robot, making it only trap and hold the designated object
- The trapping mechanism will work like that of a net - being able to encompass the object with a barrier, effectively containing it

Object Sorting:

- Be able to move the object to a designated location based on what it is
- Libraries define which objects to collect using OpenCV. These objects are then sorted into separate collection areas
- Distinguish objects by looking at specific identifiers, these range from color, shape, size, and patterns on the object.

Robot Motion:

- Servo controlled wheels allow the robot to move around on level, smooth terrain
- Able to focus in on a object in order to collect it
- Releasing the object once in the correct location

User Interface

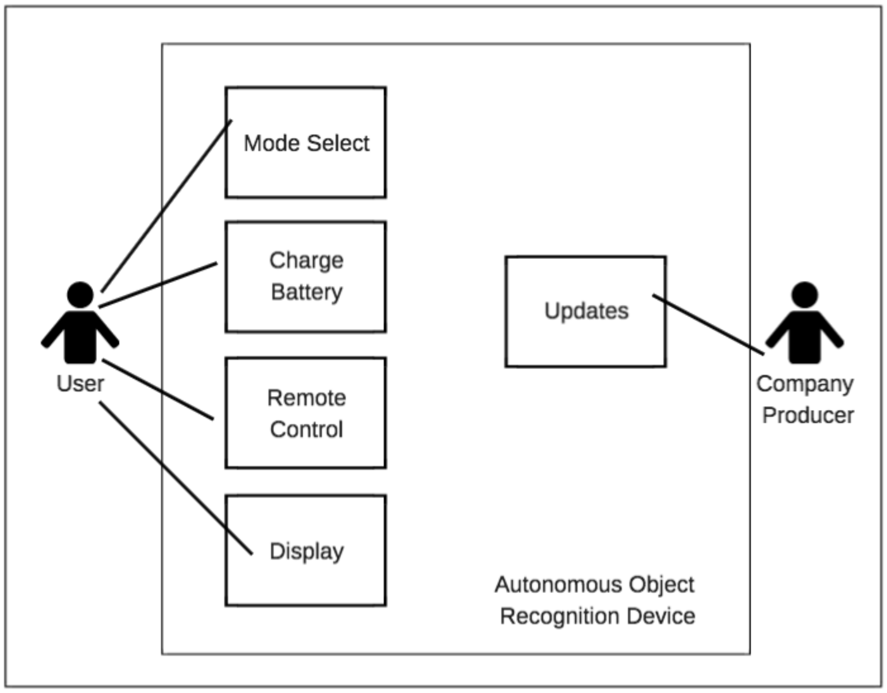


Figure 3: The UML diagram of the user interface

Mode Select:

The user will be able to select the mode between autonomous and manual mode; this is done through the user interface. In autonomous mode, the robot will move on its own and make decisions on which items should be trapped. In manual mode, the user will control the direction and trapping mechanism for the robot. Exceptions - power off, broken bluetooth/miscommunication

Charging the battery:

The user will be able to charge the battery by removing it from the robot and plugging it into an external power source. The wheel batteries can be replaced with new AA batteries.
Exceptions - power on, no external power source, damage to battery

Remote Control:

The user will be able to use manually control the robot with a bluetooth controller while in manual mode. This controller can be any sort of simple device with bluetooth capability, such as a smart phone. The user will be able to toggle between manual and autonomous mode with this connection.
Exceptions - not in manual mode, power off, no bluetooth connection

Display:

A small OLED display on the robot will provide the user with information on the task it is currently executing. This involves the current direction it is moving,

whether it has trapped an object, and the location it is looking for. It can also be used to check any errors that the robot runs into, e.g. sensor error or Raspberry Pi communication error.
Exceptions - power off, connection broken from MSP432

Updating Software:

The developers can update the software of the robot by manually connecting and downloading updated firmware to the system. For example, the developers can update which objects the robot can search (for example, new libraries to be used by OpenCV). The user will then be notified on the display interface (the OLED and the bluetooth-capable device) that the firmware of the system has been updated.
Exceptions - power off, unable to make direct connection

The user will be able to select the mode the robot is in using by sending a signal through the bluetooth communication. They will also be able to control the robot in manual mode with the use of the bluetooth controller. The user will decide which objects to pick up and where they should be returned to.

Manual Mode Options:

Modes:

- Autonomous
- Manual

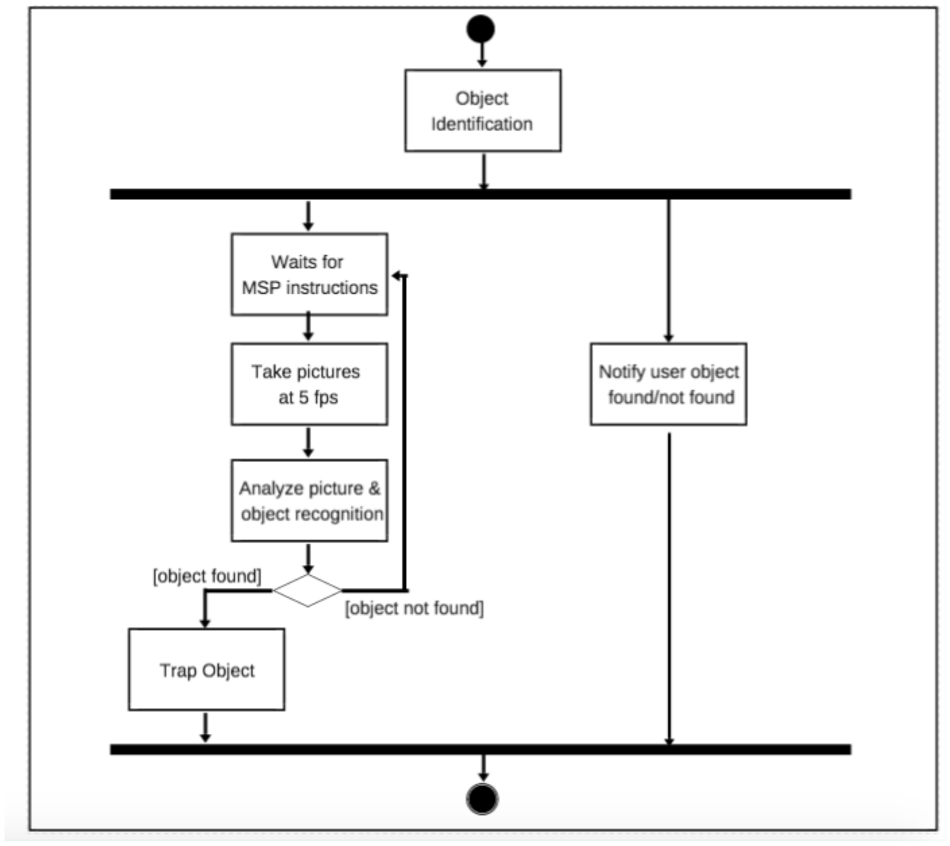
Control Inputs:

- Direction:
  - Forward
  - Backward
  - Left
  - Right
- Arm:
  - Lift
  - Lower

System Functional Specification

The system is intended to cycle through two consecutive task for collecting objects. Those tasks are (1) the ability to identify an object based on the provided library, and (2) the ability to trap and release the object at a specific location.

Object Identification - The identification of the objects will be based on the computer vision learning algorithms used by OpenCV. The robot will roam around a predefined space avoiding walls and other obstacles until the sensors for object detection are triggered. If the object is to be collected, the arm will drop, trapping the object. To identify the object after the distance sensors are triggered, the Raspberry Pi will take a burst of 5 pictures to remove false positive readings, and determine which object it is identifying.


Object Identification
Waits for MSP instructions
Take pictures at 5 fps
Analyze picture & object recognition
[object found]
[object not found]
Trap Object
Notify user object found/not found
Figure 4: The high level block diagram for the object identification functional decomposition

Object Interaction Control - There will be an arm on the robot used to manipulate its environment. This mechanism will use a capture method, where lowering the arm will trap the designated object. The robot will then transport the object to the correct drop off location. This location is predetermined by the robot using the compass sensor. In the environment we used, these locations were set as the NW and NE corners. The robot will have an arm that can be lowered and lifted using a servo mechanism. Using the arm mechanism and a wedge, this effectively traps the collected object, and allows for the robot to push it into the designated area.

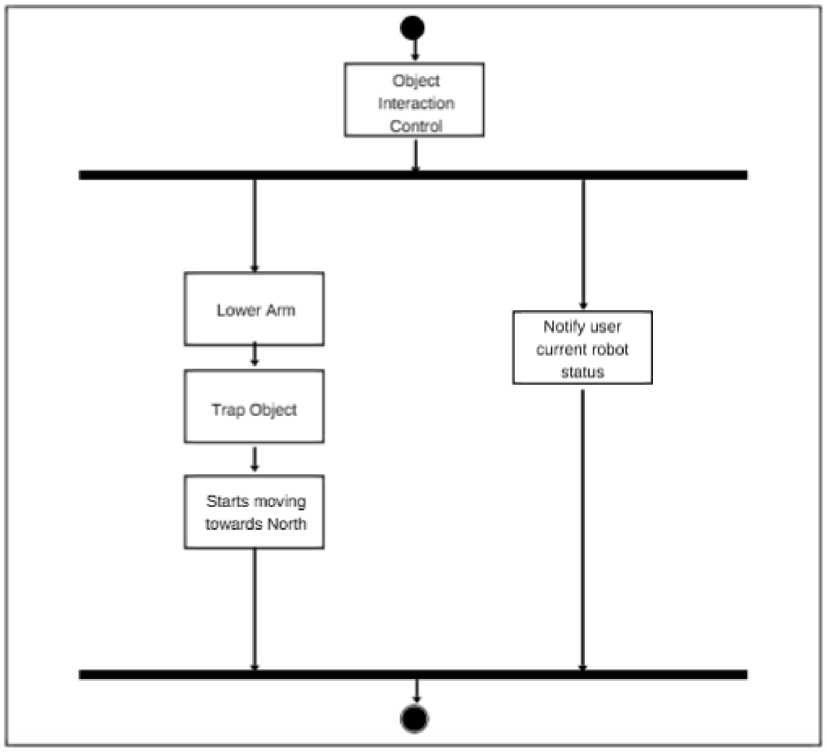

Object Interaction Control
Lower Arm
Trap Object
Starts moving towards North
Notify user current robot status

Figure 5: The functional decomposition for the object interaction control.

Obstacle Avoidance - To avoid other objects, this robot will have 5 distance sensors evenly space around the body. This will prevent it from running into walls and other environmental factors. Two are used for avoiding collisions on the sides, while another pair are on the front corners to help when turning. The final sensor is extended past the body of the robot, in the front, to prevent damage to the trapping mechanism when lowered, and to avoid obstacles from the front sooner. By using digital sensors it simplified the analysis of the incoming data. Obstacles and other objects were not detected unless it was within the range of the sensor, making the object avoid it. The data from the distance sensors will then be analyzed and used to determine whether an object is too close to the robot.

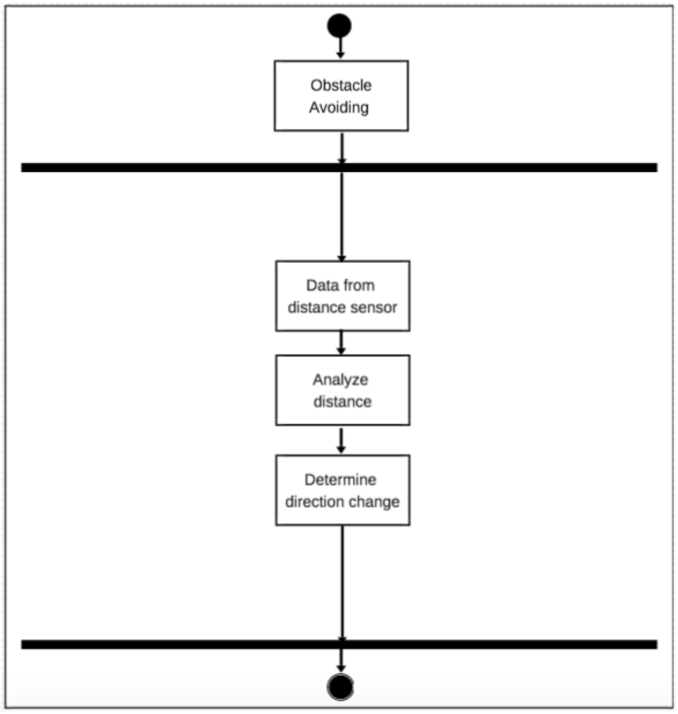


Figure 6: The functional decomposition for the obstacle avoidance.

Path Mapping - For simplicity, the robot will roam around its environment moving until it encounters an object or a wall. If it encounters a wall it will turn around and continue looking for objects to collect. If it finds an object to collect, it will trap it, and then take that object to its designated collection area. This is done using a mapping system controlled by a compass sensor. The sensor reads the current direction the robot is facing and tells it to turn either left or right in order to face the predetermined direction. In our case this direction was north. Then depending on which object was collected, the robot would either move east or west until a corner is reached, releasing the trapped object.

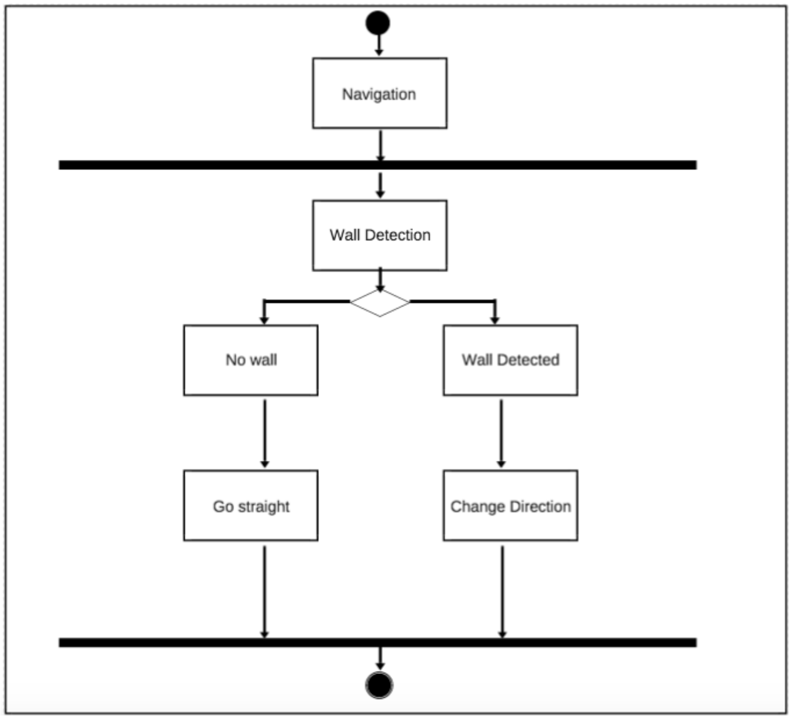


Figure 7: The functional decomposition for path mapping.

Operating Specifications

The device will operate in a standard commercial/consumer environment.

- Temperature range: 12 - 50 ℃
- Humidity up to 90% RH non-condensing
- Battery Power: 5V 10000mAh for a minimum of 8 hours fully-charged battery
- Bluetooth signal: 2.4 GHz
- Clear places with minimal impact from fog / smoke for camera vision

The system time base shall meet the following specifications:

- Temperature stability: 0-50 ℃ < 6 x 10^-6
- Aging Rate:
- 90 day < 3 x 10^-8
- 6 month < 6 x 10^-7
- 1 year < 25 x 10^-6

Reliability and Safety Specification

- Safety: UL-3111-1, IEC-1010, CSA 1010.1
- EMC: CISPR-11, IEC 801-2, -3, -4, EN50082-1
- Bluetooth safety requirement: CENELEC EN 50385:2002
- Battery requirements: IEC 62133-2
- Mean Time Between Failures: minimum of 100,000 hours

- Max object weight: 1lb

Design Procedure

Distance Sensor

The distance sensors are used to avoid obstacles and walls. Using polling in the MSP432 this changes the direction of motion when the sensors are triggered. Two sensors in the front are used to start the image processing in the Raspberry Pi.

Robot

The body of the robot was selected to be small and relatively cheap. With a smaller body it limits the objects that can be selected, however it allows the robot to access areas that a larger robot could not. It also allows smaller servos to be used, which cuts the cost for the overall project by a lot. The body had to be modified in order to allow the MSP432 and Raspberry Pi to be mounted properly with all of the other peripherals.

Servos

Continuous rotation servos:

These servos were selected for the wheel motors because they can work independently from each other using just a PWM input, controlling the speed and direction of rotation by changing the duty cycle of the PWM. This allows for rotating in place and greater control of the speed of the robot.

Arm servos:

The arm servos are used to raise and lower the trap, holding the desired object in place until it is taken to the desired location. These servos were chosen for their small form factor, low power consumption, and subsequently the low torque output.

Battery Pack

Controller and Peripheral battery pack:

This battery pack was chosen for its 2 usb output hubs, which output 5V at 1A each. This is the ideal input for both the MSP432 and Raspberry Pi boards that were used for this project.

Wheel battery pack:

The wheel battery pack was used to reduce current draw through the MSP432. This battery uses 4 AA batteries which can be replaced easily if the wheels start moving sluggishly or are nonresponsive.

OLED

OLED is used to display the current status of the robot. It is connected to the GPIO ports on the MSP. It is used mostly for debugging purpose.

MSP432

This board was used for it’s large number of GPIO pins, low power consumption, and cheap price point. Along with this the software used to debug the system is free to students. This made programming the logic for our system simple and took away many complications faced with other microcontrollers.

UART

UART communication is used to send commands between the Raspberry Pi and MSP432. This controls the transfer of when image processing should start, and which direction the robot should turn after capturing an object.

I2C

I2C is used to communicate with the OLED display. This protocol was used to free up pins and allow for greater transfer speeds.

Bluetooth

Bluetooth is used to control the robot in manual mode, which communicates with the MSP432 serially through a UART port. By using the GUI, the user can connect to the robot via bluetooth and control it’s actions, or turn it to autonomous mode.

User Input Mode

The user is able to select the mode (autonomous or manual) of the robot by interacting with the graphical user interface. This uses the bluetooth connection to change a boolean value selecting the mode currently operating in.

Compass Sensor

The compass sensor is used to find the correct corner of the searched space to drop off the collected object. This is done through the Raspberry Pi’s UART communication with the MSP432, which the compass communicates with the Raspberry Pi by I2C.

Image Capturing and Processing

The image capturing and processing are done entirely by the Raspberry Pi. So the components for this part are simply a Raspberry Pi and a camera. Although other cameras such as a webcam can be used, the camera in this project is the Raspberry Pi camera module, simply because it would work better with the Raspberry Pi.

Since the Raspberry Pi will be talking to the MSP and giving the robot directions, the very first thing that had to be considered was the speed in which data are transferred. There are several ways of doing image processing with the Raspberry Pi camera module. It can be done by taking a sequence of pictures in a short period of time or by recording a video. Although the video way is preferred because it allows the robot to be basically streaming and analyzing pictures at the same time, there is a problem with the bandwidth of the memory. It is simply not powerful enough to process the video at a rate that everything looks smooth. This can be solved by using multithreading since the Raspberry Pi has 4 processors in it, but doing so involves a certain amount of complexity. So taking a sequence of pictures at a slower and smaller frame rate like 5 frames per second was the next logical approach. Still 5 frames per second is more than what the system for this project needs.

Failure Modes Analysis/Failure Management Scheme

Table 1: Failure mode analysis

Potential Failure Mode	Potential Failure Effect	Severity (1-10)	Failure Causes	Action Recommended
Power	- Inoperable device - Bad signals	10	-Power failure -Poor connection -Not enough power	-Restart the device and program -Check connection
Connection (wiring)	-Bad signals -Wrong data	8	-Bad connection -Bad wire -Dysfunctional hardware components	-Check connection -Replace wires/components
User error	-Invalid data	5	-User not following guide line	-Reconnect the device -Read user manual more carefully
Data lost	-Serial communication lost -Didn’t write to memory correctly -Wrong data	10	-Hardware problem -Software problem	-Reset device -Debug problems -If too severe, redesign

Some of the realistic problems in this autonomous recycling robot include an assessment of the reliability of image processing and object recognition training, and a reliable power system that can provide the entire system with enough current. Accuracy in which objects are recognized can be improved by having more pictures in the database for the machine to generate a reliable classifier. Failure of a power system is overcome by using more reliable battery pack. Resetting the device can restart the system and can sometime clear out system failure. This device is to be kept under normal environmental condition to prevent hardware failure.

Software Implementation

Software Block Diagram

The diagram below shows the high level functionality of the software used for this project. The two main branches from `Mode Select` show the branches of manual control versus autonomous motion. The manual side simply waits for input from the user and follows the instructions provided. The autonomous side uses all of the sensors and direction inputs on the robot in order to find objects and move them to the correct destination.

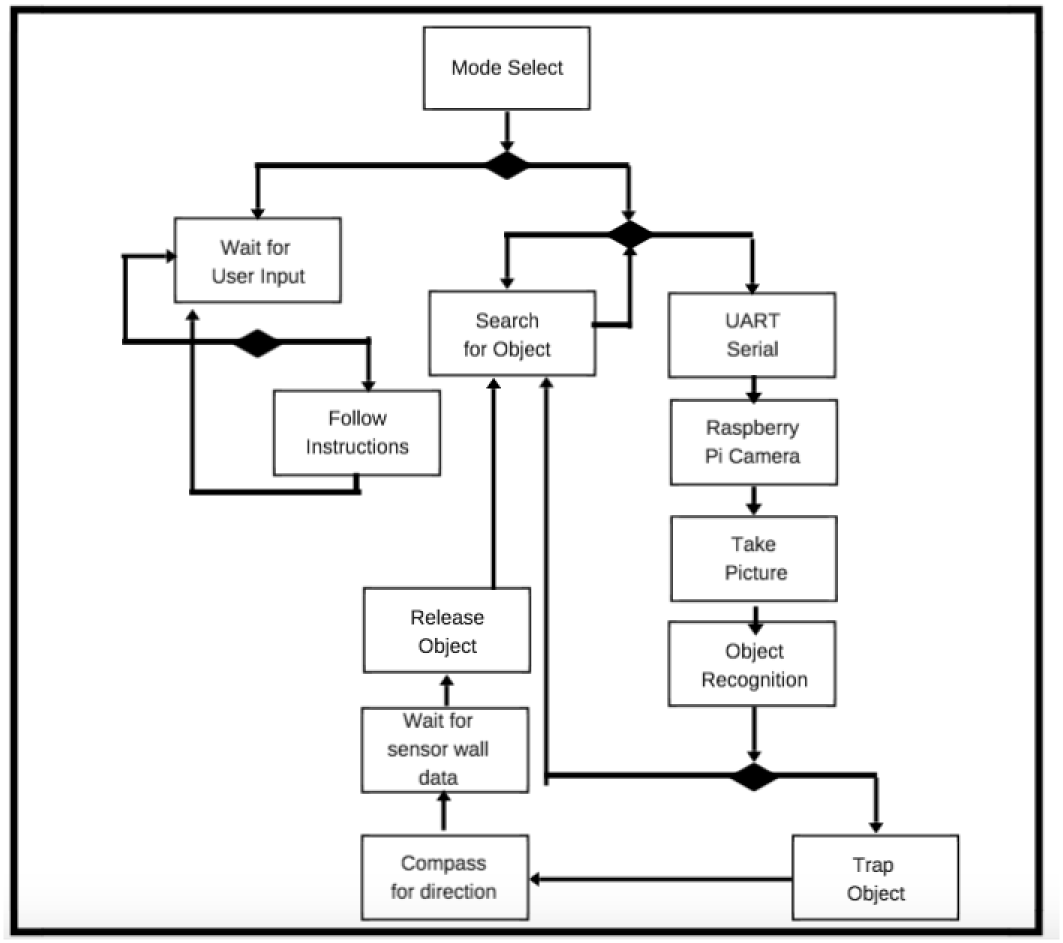


Figure 8: Software block diagram for the autonomous robot

Functional Decomposition

The diagram below shows the functional decomposition for this project. Split between the main controller MSP 432 and the secondary controller Raspberry Pi, it lists the responsibilities and peripherals associated with each.

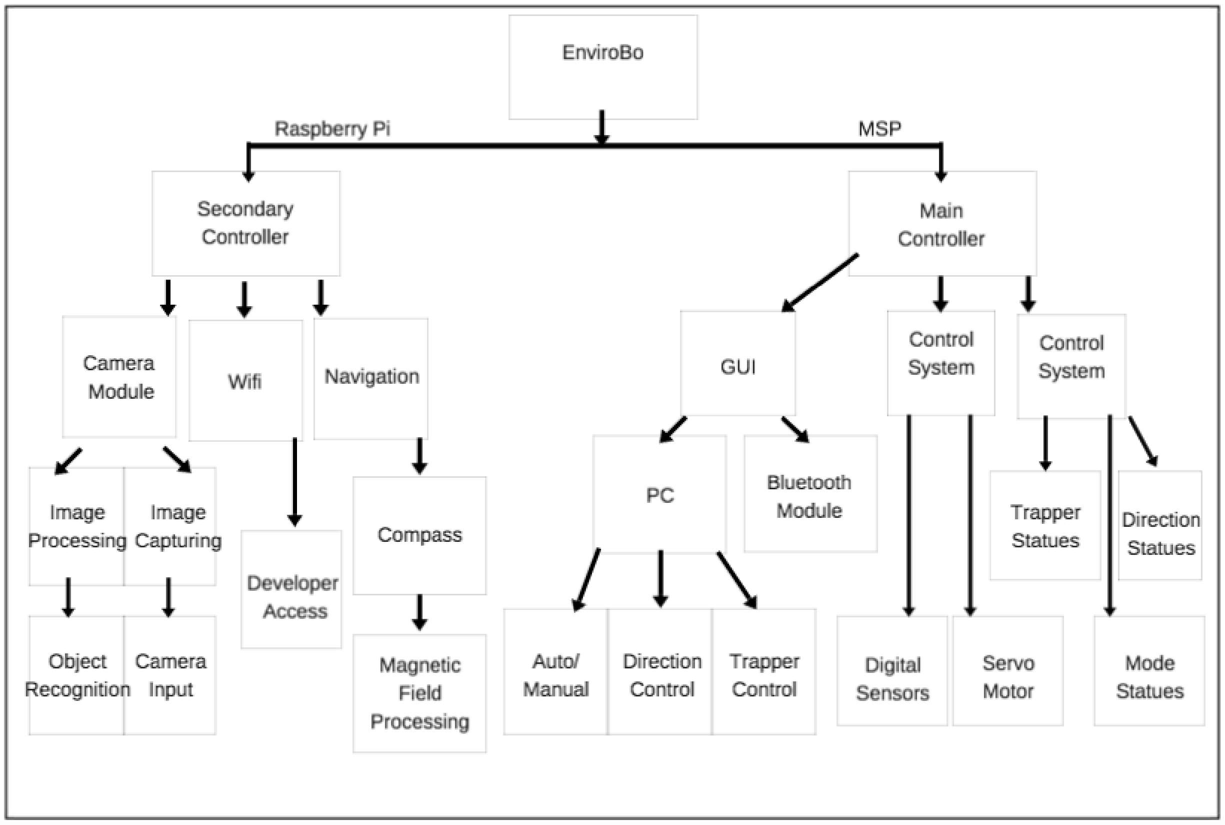


Figure 9: The functional decomposition on the autonomous recycling robot

Bluetooth Module and Graphical User Interface (GUI)

The bluetooth module is used to interact with a user's personal computer. The bluetooth is configured with MSP432 through UART. In the user side, a GUI developed in Matlab is provided for the user to choose between autonomous and manual mode. When the autonomous mode is selected, the robot will automatically search for an object. The GUI also reports about the direction status, the trapper status, the bluetooth connection status, and the mode status. When the manual mode is selected, the user will be able to control the following:

Direction control: forward, left, stop, right, and backwards
Trapper control: lower and lift trapper
Bluetooth connection: connect or disconnect

The GUI also reports about the direction status, the trapper status, the bluetooth connection status, and the mode status.

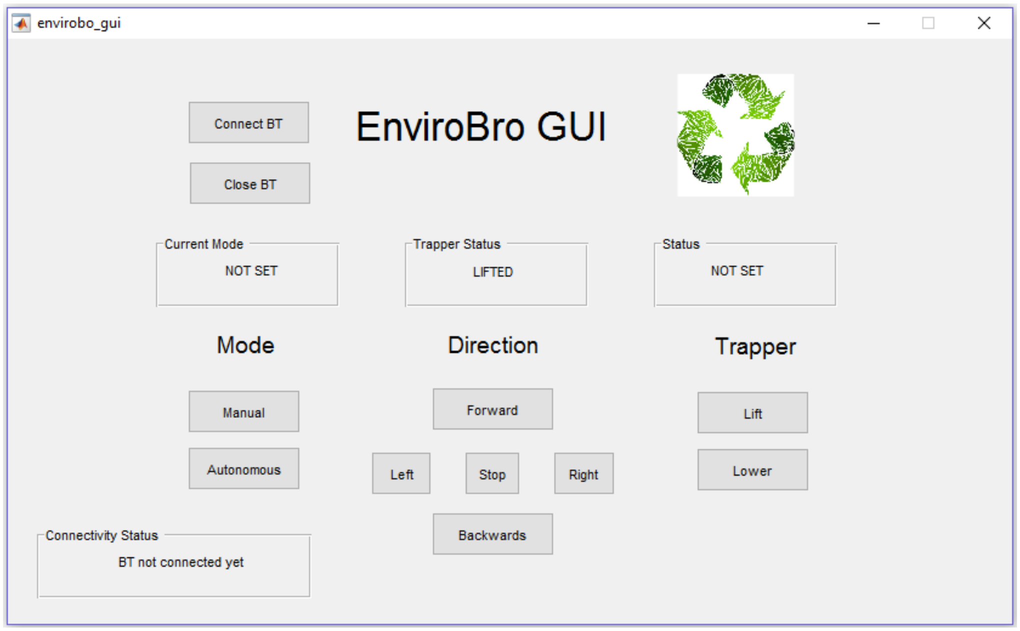


Figure 10: The GUI display for the system

<u>OLED</u>
OLED is used to display the current status of the robot. It is communicating with the MSP432 firmware system through I2C. One of the software components of OLED is the character library that has to be created to display certain characters on the OLED. The steps for configuring and using the OLED are:

- Access device address 0x3C
- Write to the the address to clear data
- Send character bytes to the character register

<u>MSP432 and Raspberry Pi Intercommunication</u>
UART is used to communicate between the MSP432 and Raspberry Pi, transferring information for object recognition and compass information. The Raspberry Pi waits for commands from the master controller, MSP432, which sends a signal to start the image processing when there is an object in front of the distance sensors. It continues exchanging data to determine if the object should be trapped or not. After trapping an object the UART connection is used to start the compass program, giving the robot a heading for finding the location to drop off the object. The following is the list of commands that are sent back and forth between Raspberry Pi and MSP432:

Autonomous Mode

- ‘g’ -- MSP432 to Raspberry Pi: sent when the system detects an obstacle for the Raspberry Pi to be analyzed
- ‘F’ -- Raspberry Pi to MSP432: sent if object is found and the object is ready to be trapped
- ‘M’ -- Raspberry Pi to MSP432: sent if object is not found and the robot will continue searching for objects
- ‘t’ -- MSP432 to Raspberry Pi: sent if object is trapped and the navigation system will take over
- ‘s’ -- Raspberry Pi to MSP432: sent if the robot points to the northern wall and get ready to go straight
- ‘l’ -- Raspberry Pi to MSP432: sent if the robot is not pointing to the northern wall and turn the robot left until an ‘s’ byte is sent to MSP432
- ‘r’ -- Raspberry Pi to MSP432: sent if the robot is not pointing to the northern wall and turn the robot right until an ‘s’ byte is sent to MSP432

Manual Mode

- ‘a’ -- switch to autonomous mode
- ‘m’ -- switch to manual mode
- ‘f’ -- go forward
- ‘b’ -- go backwards
- ‘l’ -- go left
- ‘r’ -- go right
- ‘t’ -- trap object
- ‘y’ -- lift trapper up

<u>MSP432</u>

The MSP432P401R LaunchPad made by Texas Instruments is used as the main controller for this project. It controls all of the logic for robot motion, the trapping mechanism, and triggering the object recognition of the secondary controller. Polling is used for updating the data received from the peripherals, as the system works in a single cycle order and does not require high speed updating of input variables. Several boolean variables are used for checking the state of each sensor and the current mode of the robot is also set by a boolean variable.

```
main() {
        setup(); //sets all pins being used to the correct configuration

        //bool variables for holding the state of the digital sensor input
        bool right;
        bool left;
        bool rFront;
```

```
    bool lFront;
    bool brFront;
    bool blFront;
    bool ctFront;

    while(1) {
        if (manualMode) {
            manual(); //function to control the robot while in manual mode
        } else {
            //logic for autonomous control
            update variables with current state of the sensor;
            if(object detected) {
                start communication with Raspberry Pi;
                wait for response;
                move according to input received;
            } else if (other sensors triggered) {
                move away from the direction of the triggered sensor;
            }
            .
            .
            .
            //this logic is copied for other combinations of sensors to create
            //object avoidance
        }
    }
}
```

<u>I2C and Compass Sensor</u>

The compass sensor is a central part to the navigation system used by the robot. It communicates direction data to the Raspberry Pi using I2C communication protocol. This data is used when an object has just been trapped and the direction must be determined to find the designated location to drop off the collected object. The compass sensor is configured through the i2c-dev Python libraries in Raspberry Pi. The address of the device is 0x1E. Before getting the data from the compass, the compass' measurement mode has to be set to be continuous. To get the data out of the compass, the Raspberry Pi has to access register 0x03 and read six bytes from the aforementioned register. The register contains the most significant bytes and the least significant bytes of the x, y, and z values. The values obtained from the compass sensor are measured in microtesla. What the values represent are the projections of the strongest magnetic field vector in the surrounding environment. We can then get the heading of the sensor by calculating the angle between the x vector and the y vector by using the arctangent function. Magnetic declination also comes into play. We had to add an offset value of 16^{O} to the heading because Seattle's magnetic tendency is off by 16^{O} from the true north (north pole). Below is the pseudo code of how the magnetic field is used.

```
main() {
    setupCompass(0); //Initiates I2C and set compass to be in continuous mode

    // to hold the six values (8-bit, MSB and LSB for x, y, and z)
    int[] array_of_six;
    // sees an object is trapped
    bool trapper;
    // sees if the system is pointing north
    bool correct_heading;
    while(1) {
        if (object not trapped) {
            run python2's image processing algorithm;
            // the above program will exit if an object is trapped
            trapper = 1;
        } else if (object is trapped and robot not pointing north) {
            read from MSP432;
            if (read == 't') {
                tell MSP to turn right or left
            } (read == 's') {
                tell MSP to go straight
            }
        }  else if (object points north) {
            Wait until MSP432 detects a wall;
            if (object is faces) {
                turn right;
            } else {
                turn left;
            }
            wait until release signal, then break;
        }
    }
}
```

Camera and Image Capturing

Camera is initialized using the OpenCV library and the Raspberry Pi Camera module library. It has a resolution of 320 * 240 pixels and captures .jpeg images in a 5 pictures per second frame rate by calling the imwrite() function. Pictures are converted into a numpy array for OpenCV to process by using the numpy.fromstring() and the .imdecode() functions.

Object Recognition

The object recognition requires the Open Source Computer Vision library (OpenCV) with the Haar feature-based cascade classifier. The cascade classifier is a machine learning algorithm that uses many images to train a classifier that can later be used to detect that object. The general idea is to collect an image database and do the following steps so that features shown in Figure 11 can be learned and collected into an XML file.

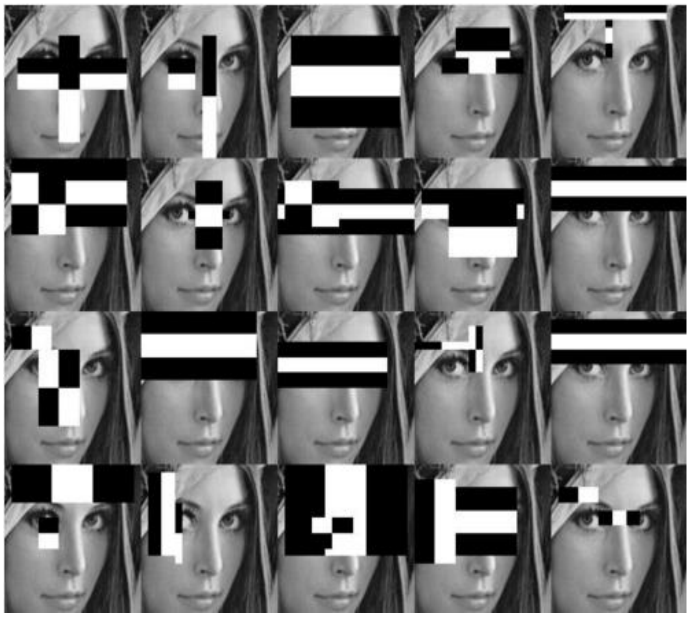


Figure 11: An example of features trained in the facial recognition XML

The first step is to collect an image database that includes a set of positive images which is the object to be recognized, and a set of negatives images which corresponds to the background or anything that is not the target of the recognition. Having more number of positive and negative images will normally cause a more accurate classifier. The second step is to arrange negative images and put them into a folder and run a batch file that will be used for training the classifier. The third step is to crop and mark positive images by using the image clipper. The next step is the actual Haar training which is packing the object images into a vector-file. It will execute a number of stages until the ratio of misclassified samples to candidate samples in lower than a certain threshold value, meaning that when the training has generated enough positive images. Depending on the number of pictures that was feed to the program, the training process might take up to a week. The final step is to combine all created stages or classifiers into a single XML file which will be the final file for the cascade of Haar-like classifiers.

msp_rasp_1:

```
while(1) {
        Wait for MSP to send flag
```

```
    if (nothing not trapped) {
        initialized resolution, frame rate of the camera
        start preview window
        while (haven't taken 10 pictures) {
            capture sequence of pictures
            for (each image captured, do image processing) {
                load cascade file for detecting target
                look for clocks and faces (or bottle and banana) in the image
                if (first image captured) {
                    compare which one is closer
                    generate a file with which object was detected
                }
                else { //image processing for the rest of the 9 pictures
                draw boxes around the object found
                increment the go count by 1
                }
            }
            save these 10 images into 10 different .jpeg file
            check the number of go count
            if (go > 6) {
            Send lower-trap flag to MSP
            } else
            Send miss flag to MSP
            stop preview window
        }
        reset all flags and counts to 0
    } else //When flag is "t"
    break
}
```

Hardware Implementation

Hardware Block Diagram

Figure 12 shows the high level functionality of the hardware used in this project. The MSP432 and Raspberry Pi communicate through a UART connection, creating system used to control the peripherals and process their input data.

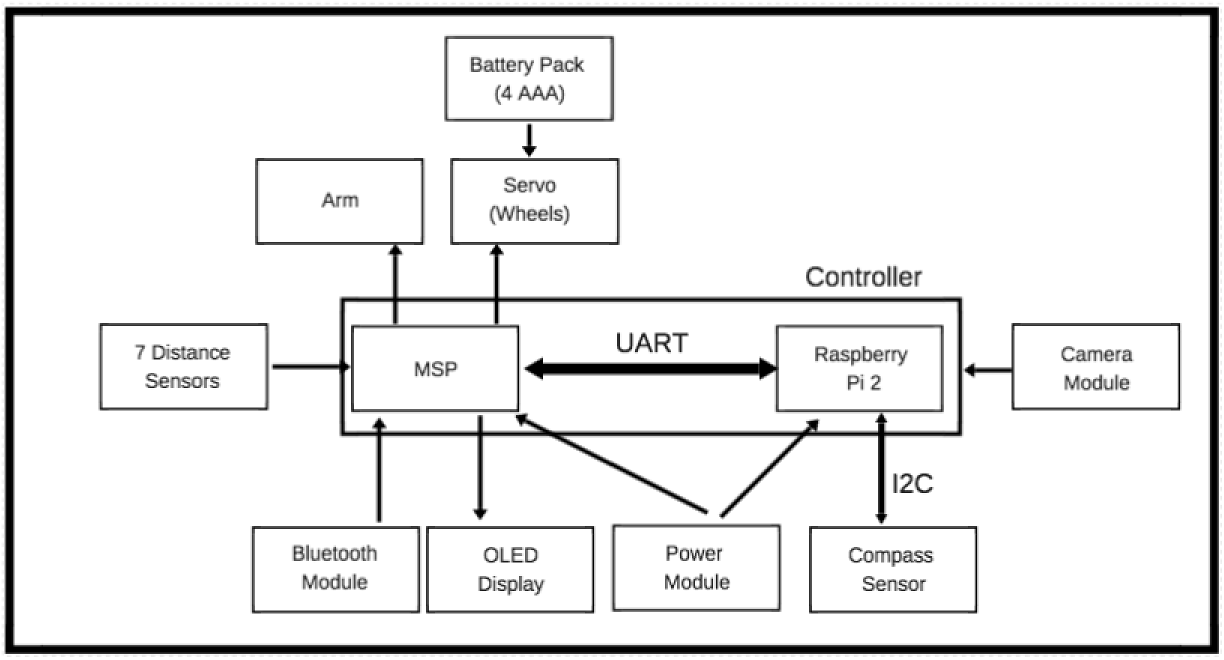


Figure 12: The hardware block diagram for the entire autonomous recycling robot system

<u>Distance Sensors</u>
The digital sensor we used is the Sharp GP2Y0D810Z0F digital sensor that can sense an obstacle of about 10 cm. Each sensor required a five volt input, with a capacitor connected between ground and VCC. This created less strain on the system and provided more stable output from the sensors. This is because the sensors draw large amounts of current in short bursts, so by having a 10 µF capacitor this reduced the initial strain on the system.

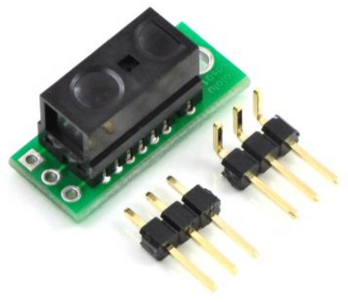


Figure 13: The Sharp GP2Y0D810Z0F digital sensor

The distance sensors use GPIOs as inputs for the MSP432 board. Using seven sensors required seven separate pin assignments. The following pins were used as inputs for the digital distance sensors for this system.
GPIO Pin Assignments:

- Right sensor = P2.3

- Left sensor = P5.6
- Right front sensor = P6.6
- Left front sensor = P6.7
- Center top front sensor = P5.0
- Bottom right front sensor = P5.7
- Bottom left front sensor = P5.2

<u>Robot - Chassis</u>
The body used for this project was a custom built platform to hold all peripheral devices and controller systems. On the bottom layer the two continuous rotation servos were mounted and used as motors for the wheels on the robot. Added underneath that was a battery pack to power those motors. A platform was added above this, out of the way of the wheels, in order to mount the necessary devices in order to control the robot. All attachments were done using corner brackets for sensor devices, and the control boards were mounted using spacers and screwed in place. A picture of this can be seen in the results section of this presentation.

<u>Servos</u>
Continuous Rotation Servos:
These servos are used to control the motion of the robot, forwards, backwards, left, and right. By configuring a PWM's duty cycle for each servo, with a refresh rate of 50Hz, the wheel is able to turn at different speeds, either clockwise or counterclockwise. Forwards and backwards motion is set by having the duty cycle of each wheel be equal in magnitude, but opposite in direction. For example forward motion was done by setting the left wheel's duty cycle to 8% and the right wheel to 3%, with no motion having a duty cycle set to 5%.
PWM Pin Assignment:

- Left Wheel = P2.4
- Right Wheel = P2.5

Figure 14: FeeTech FS5103R continuous servo

Arm Servos:

These servos are used to control the trapping mechanism that contains our collected objects for transportation. By changing the duty cycle of these PWM's this changes the position that the servo is at. There were two positions that these servos needed to be in, up and lowered. The servos are in the up position when looking for objects or releasing an object, and they are in the down position when an object is trapped.

PWM Pin Assignment:

- Left servo = P2.6
- Right servo = P2.7

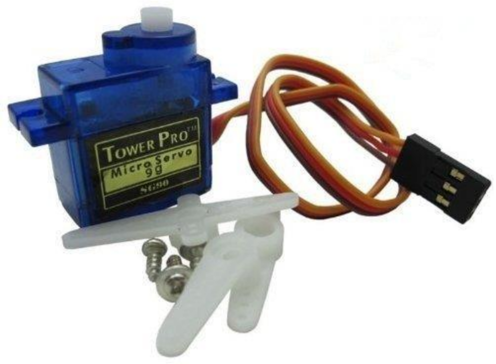


Figure 15: TowerPro SG90 9G Mini Servo

<u>Battery Pack</u>

Controller/Peripheral Battery Pack:

This battery pack is used to power both the controller boards, distance sensors, compass sensor, and camera. The peripheral sensors are powered through the board's power pins. This battery is rechargeable and provides 5V and 1A to both systems.

Figure 16: USB Battery Pack - 10000mAh - 2 x 5V @ 2A

Wheel Servo Battery Pack:

This battery pack is used to separately power the wheels as they require a large amount of power to run continuously. This was accomplished by adding a second battery pack to the underside of the robot, using 4 AA batteries. This provides an average voltage of 5V. These were then connected to the VCC and ground terminals of each servo, while also connecting it to a common ground with the MSP432.

"To comply with USB 2.0 voltage requirements only AAA batteries with max voltage of 1.25V can be used."

Figure 17: The Atmel 4 AAA battery pack

<u>OLED</u>

The OLED we used is the Adafruit SSD1306 type. The OLED is accessed through I2C via the SCL and SDA pins. The SDA pin of the MSP432 (pin 1.6) is connected to the SDA pin of the OLED and so are the SCL pins (pin 1.7 for MSP). The clock speed is configured at about 300 kHz. This module needs a minimum 3.3 V power supply.

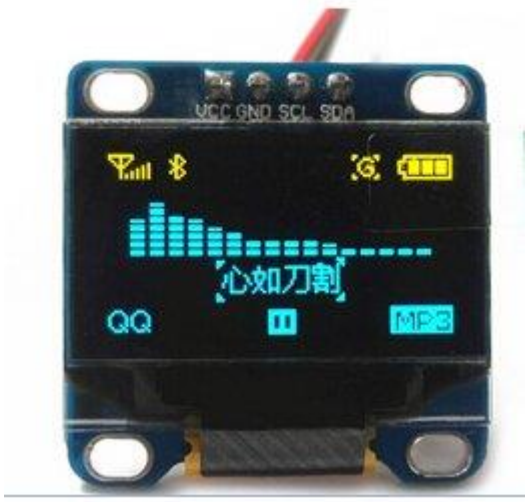


Figure 18: The hardware pinout for the OLED display.

<u>MSP432 and Raspberry Pi Intercommunication(UART)</u>

Communication to the Raspberry Pi via a serial communications port was done using UART. MSP432 has a total of four UART channels, and the Raspberry Pi has a total of one UART channel. In this design, we used UART3 of MSP432. The RX port of the MSP (pin 9.6) is connected to the TX port of the Raspberry Pi (pin 10) whereas the TX (pin 9.7) port of the MSP is connected to the RX port (pin 8) of the Raspberry Pi. The voltage level for the communication is 3.3 V for high and 0 V for low. A detailed list of flags for communication between the Raspberry Pi and MSP is documented under the functional decomposition MSP432 and Raspberry Pi Intercommunication section of this report.

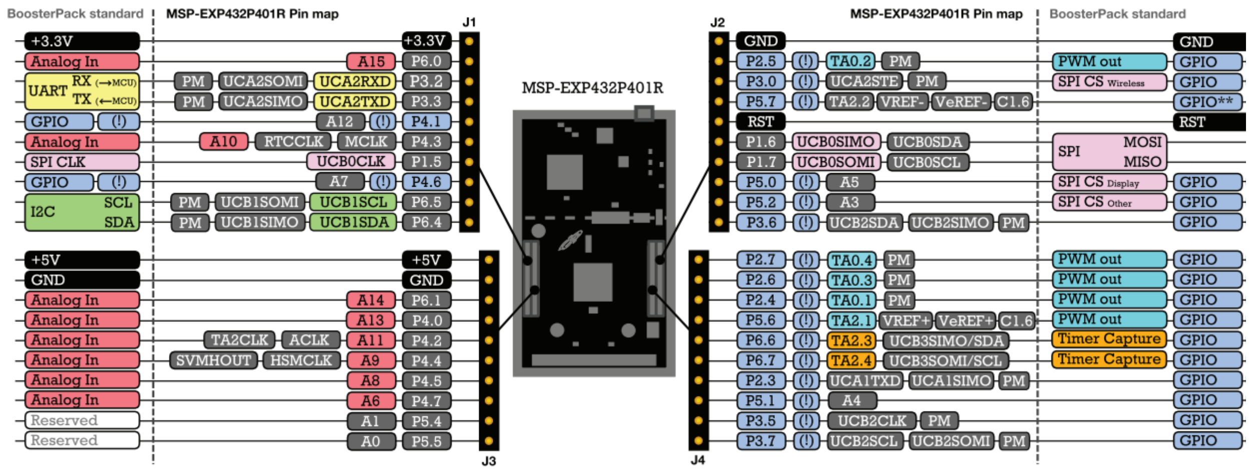
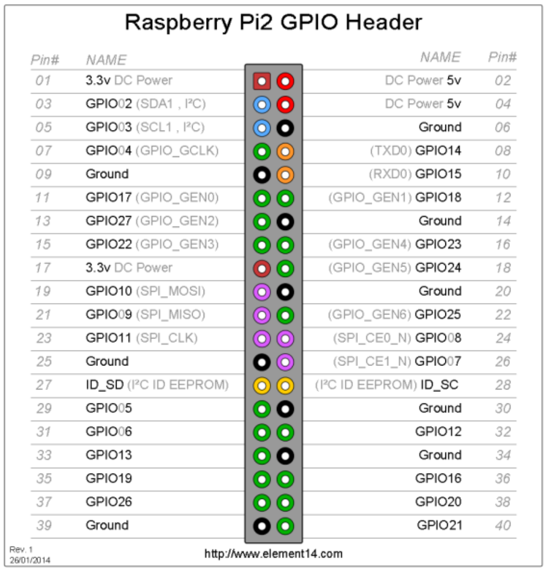


Figure 19: The hardware pinout for the MSP432.



Raspberry Pi2 GPIO Header

Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I²C)		DC Power 5v	04
05	GPIO03 (SCL1 , I²C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I²C ID EEPROM)		(I²C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

Rev. 1
26/01/2014
http://www.element14.com

Figure 20: The hardware pinout for the Raspberry Pi 2.

Bluetooth Module

The Bluetooth module is configured through UART in MSP432. The TX pin of the Bluetooth module is connected to the RX (pin 3.2) pin of UART2 in MSP432. And, the RX pin of the Bluetooth module is connected to the TX (pin 3.3) pin of UART2 in MSP432. The voltage level for the UART protocol is 3.3 V. The 3.3 V power coming from MSP432 is connected to the V_33 pin of the Bluetooth module.

Figure 21: The hardware pinout for the bluetooth module.

Compass Sensor

We used the GY-85 IMU sensor. The IMU sensor composes of a magnetometer, accelerometer, and gyrometer. The reason we chose this sensor is that we can save on the usage of excessive pin numbers but still able to connect to multiple sensors. We finally used the magnetometer and accelerometer for our system. The accelerometer is used to level the sensors so that the magnetometer gives a consistent reading. The SCL pin of the IMU sensor is connected to the SCL (pin 1.7) of MSP432, and the SDA pin of the IMU sensor is connected to the SDA (pin 1.6) of MSP432. The IMU sensor is powered by a 5 V power supply from MSP432.

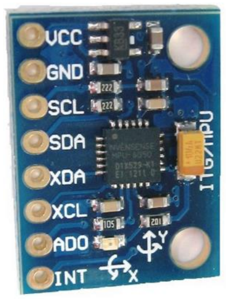


Figure 22: The hardware pinout for the compass sensor.

TEST PLAN

The testing for this project was split into three parts, the mechanics, intercommunication, and object recognition, and then finally being tested as a whole.

Mechanics

The mechanics of this system includes the motion of the robot, the trapping mechanism, and the distance sensors. Testing for this part of the project was done by first setting up the wheels to function correctly with a PWM on the MSP432 board. This involved making sure that they were in sync with each other for moving in a straight line, and turning precisely. Once configured the distance sensors were all tested for accuracy and correct functionality, having an output of 1 when not triggered and 0 when triggered. These were then mounted to the system with the MSP432. When a sensor is triggered, the logic of the system forces the robot to turn away from that direction. This was tested by changing the length that the PWM's duty cycle is set to the value for turning. This allow for greater control on how well the robot could move away from objects and walls. Sensor placement was also tested for optimal blindspots. Since this robot does not move backwards autonomously, there are no sensors in the back, focusing mainly on avoidance from the front. The center front sensor does the greatest job in avoiding objects as it is the first sensor to be triggered when moving towards a wall.
The arm servos were simple to add, after having to configure the wheel servos. All that was needed was to properly attach them in a position that did not interfere with any distance sensors and allowed for consistent trapping of objects. To set the positions for the up and down position it took testing with different duty cycles until the desired angles were met.

The power system was also tested for these devices. A rechargeable battery with two usb ports was used to power the MSP432 and Raspberry Pi, while a separate battery was used to power the wheel servos. This required testing to make sure input signals were within range of the power voltages, requiring a common ground throughout the entire system.

Intercommunication

The testing of the intercommunications involves configuring UART, I2C, and bluetooth communication. The UART for both the MSP432 and Raspberry Pi were first tested separately using a usb connection with a communication program, CoolTerm. This involved checking for the correct baud rate and making sure the buffer does not overflow. Once each device had UART running separately with the same baud rate they were connected together and tested by passing characters back and forth. The I2C protocol was used for the OLED display on the MSP432. This required functions to initialize, write data to, and clear data from the screen. Testing this required debugging the protocol itself, while checking that register values had the correct numbers stored at the right times. Finally the bluetooth connection was tested using the UART connection previously established. This allowed for configuration of the bluetooth module, and testing with other bluetooth enabled devices. Testing for the correct connection involved sending characters from a connected device and seeing them on the OLED attached to the MSP432. After initialization, connecting the output to a UART channel on the MSP432 was the next step. This was relatively simple as only the baud rate needed to be configured correctly.

Object Recognition

The testing for object recognition includes testing the functionality of OpenCV, the communication between Raspberry Pi and the camera module, and a WiFi connection. To start configuring the Raspberry Pi to run OpenCV must be tested for correct installation. After installing the next step is to initialize the camera module. After configuration of the camera for use with the system, compiling the code, and detecting objects, using built in libraries is the first goal and then importing our own is the next. After having OpenCV correctly running and analyzing objects from the provided libraries the next step is wireless communication. By having a WiFi module this allows greater mobility of our system. This was tested by setting a static ip address for the board and reconnecting to it after restarting the system. With the hardware setup, testing the object recognition code was next. This involved writing code to determine which object has been detected, and output commands that would be used to communicate with the MSP432.

Modeling Plan

An initial plan for modeling the communication between MSP432 and the Raspberry Pi.

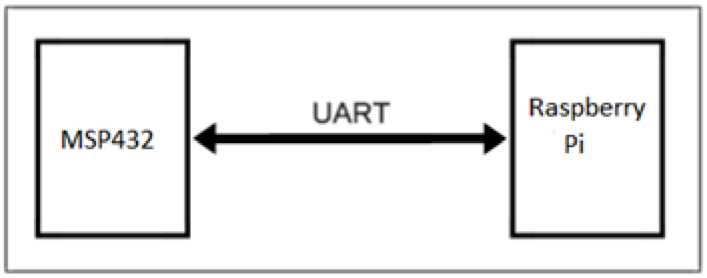


Figure 23: UART communication model

TEST SPECIFICATION

Distance Sensor

The sensor should be triggered when an object is within 10cm of the sensor. It has to be property powered, and for added stability a capacitor is added between ground and VCC.

Robot

Make sure all wheel servos, digital distance sensors, and arm servos are properly attached. A neutral environment is setup with a flat surface and nothing for the robot to get stuck on. The balance of the robot is so that the wheel have enough traction to move evenly and smoothly.

Servo

The PWM input pins are securely attached to the input wire of the servo. The power to each servo is a constant, steady 5V. Common ground is used throughout the servos and the MSP432.

Battery Pack

The battery pack is fully charged and properly connected to the power input of the control boards. The battery pack for the wheels are properly connect, while ground is connected to the ground of the entire system.

OLED

The OLED screen will be another way to debug our measurement phases. This will need to be wired and powered properly in the circuit. Once this is done, then we will need to test how many characters can be written per line, how to choose which line to write to, what happens when too many characters are sent to the display, and how to clear the display.

MSP and Raspberry Pi Intercommunication (UART)

The UART is used to allow communication between the Raspberry Pi and the MSP. Data is exchanged between these devices through the RX and TX ports. The baud rate and FTDI USB to UART converter will have to be tested in order to have the MSP communicate correctly with the Raspberry Pi. This involves making sure all connections are good and to the right pins. The baud rate will require testing and checking, to make sure the data if transferred wholly and correctly.

Bluetooth and User Input Mode

The bluetooth connection is properly setup to communicate with the MSP432 through UART. The GUI properly sends commands to the bluetooth module that does not overflow the UART buffer.

Compass Sensor and I2C

The compass sensor is properly aligned and level, while being far enough away from metal objects to not interfere with magnetic readings for direction. The I2C communication is initialized properly and connected to the correct ports of the Raspberry Pi.

Camera

The Raspberry Pi 2 camera module should be able to capture image within a certain amount of time frame and record a video that can be saved into a .jpeg file.

Object Recognition

The cascade classifier should be able to identify an object based on the image that is feed in. It should correctly draw a box around the detected object.

TEST CASES

UART

To make sure the data is transferred correctly, the baud rate needs to be tested. One way to test this is to make sure that if we send a character from Raspberry Pi or MSP to the PC by using the application Coolterm, the application on PC can receive that character and display that character on the serial terminal.

Display

Hardcoding string to display on the OLED is the most obvious way to test the display. Clearing the display and changing the start point for printing is tested by printing test strings to the display.

Distance Sensors

Testing the distances that the sensors trigger at, and how quickly the output changes when triggered. Testing repeated triggering and the resolution of the changes in output.

Servos

Continuous rotation servo:

Testing the different speeds produced with different duty cycles. The change that input voltages has on the servo's rotation speed. Testing the alignment of the wheels when moving forwards and backwards. Testing the are stopped when told to stop, no rotation.

Arm servo:

Testing the different angles when raised and lowered. The arm should be fully lowered in the trapped state. The arm should not interfere with distance sensors when raised or lowered.

Compass Sensor

Testing the effect that metal has on the reading. Turning the robot and comparing it to known directions, making sure that north actually points north. Test the effects of rotating the compass off axis, for example turning it sideways instead of rotating in the same plane.

Bluetooth Module

The bluetooth module will be test for max distance for a stable connection, how much data can be sent at a time before the UART communication buffer overflows, and the correct outputs from specific input arguments.

Camera

The Raspberry Pi camera module is tested using its own library. We started by taking an image and saved that image into a desired directory. Then it was tested by running a real-time streaming video. A preview screen was added so that we can actually see what the camera is pointing at and therefore position of the camera can be fixed accordingly.

Object Recognition

The facial recognition was used first for testing because the xml database came with the library when the OpenCV was installed, therefore, it provides a more accurate set of features in identifying the object than any other cascade classifiers.

PRESENTATION, DISCUSSION, AND ANALYSIS OF THE RESULTS

After thoroughly debugging and testing our design, the results of this lab was a system that could successfully identify an object, trap it, and move it to a designated location. There were still some bugs with our system that need to be address, but for the majority of tests the robot behaves as required, as well as further implementations to improve the functionality and usefulness of this project.

As shown in the figure 24 below, the major features of this autonomous recycling robot include a Raspberry Pi 2, MSP432, distance sensors, an OLED display, a bluetooth module, an object trapping mechanism, and a compass sensor.

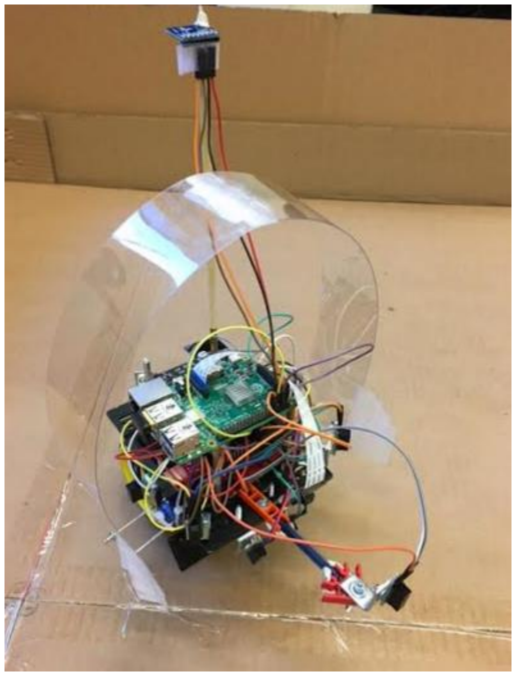


Figure 24: The final product of the autonomous recycling robot

Object Recognition

Once the distance sensors in the front of the robot detects something if front of it, MSP is able to tell the Raspberry Pi to initialize the camera and start taking pictures. Image recognition accuracy, as discussed earlier in this report, highly depends on the amount of data in the database during the training of the cascade classifier, therefore, does not function perfectly whenever the target object is in the image. The autonomous is able to identify faces, bottles, and banana at some distance away from the objects as shown in Figure 25 and Figure 26.

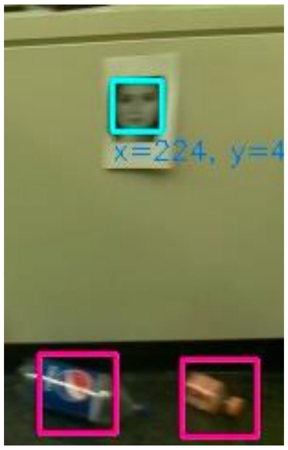


Figure 25: Result of facial and bottle recognition

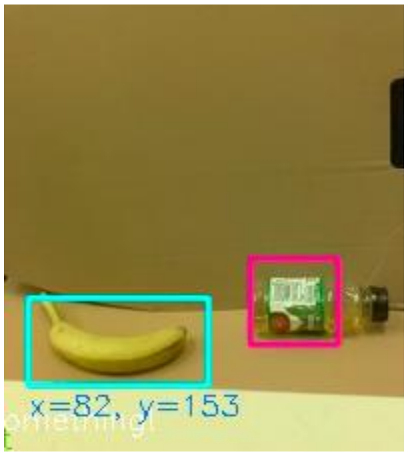


Figure 26: Result of banana and bottle recognition with little background noise

The image processing part can also determine where the target object is relative to the robot. As shown in Figure 27 below, when the object is to the right/left of the camera, the Raspberry Pi will tell the MSP go left/right, whereas when the object is just in the middle of the picture, a trap signal will be send to the MSP telling it to lower its arm and trap the object in front .

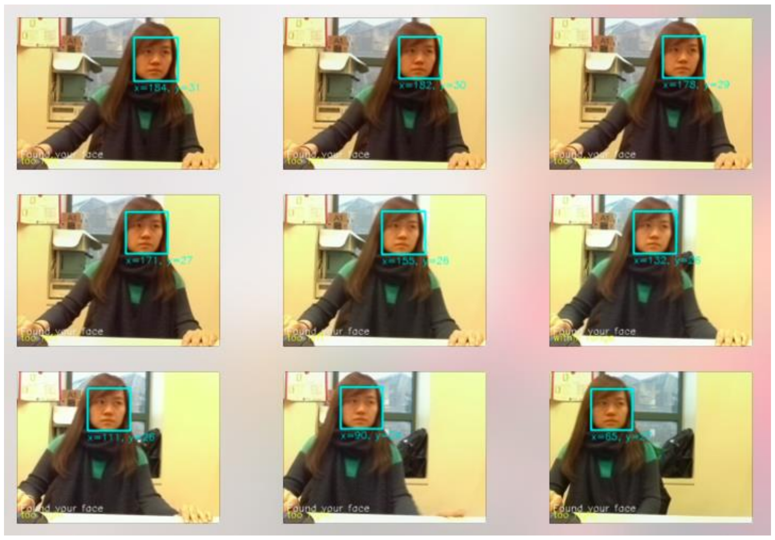


Figure 27: An example of sending position flags (left, right, in range) to the robot

Environment

An arena shown in Figure 28 had to be built because of two reasons. First, the cardboard eliminates some of the background noise that will be visible to the camera, which helps the accuracy in identifying objects. It also provides a better surface for the robot to drive on since the friction force of the cardboard is smaller than the floor. Corners had to be predefined as well to be the recycling bins for the robot to put the objects to. The locations are determined by the compass sensor that knows where the relative north is in the arena.

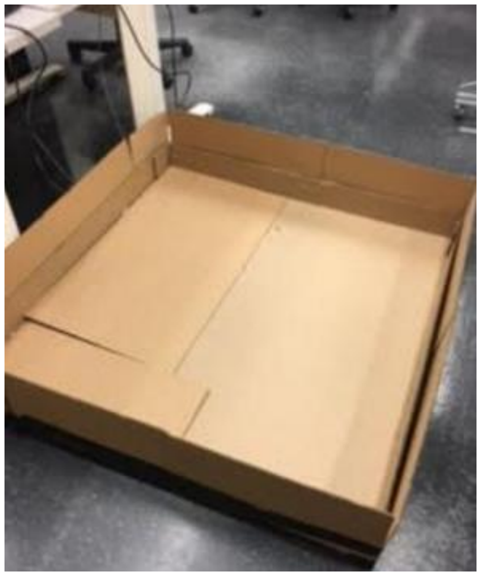
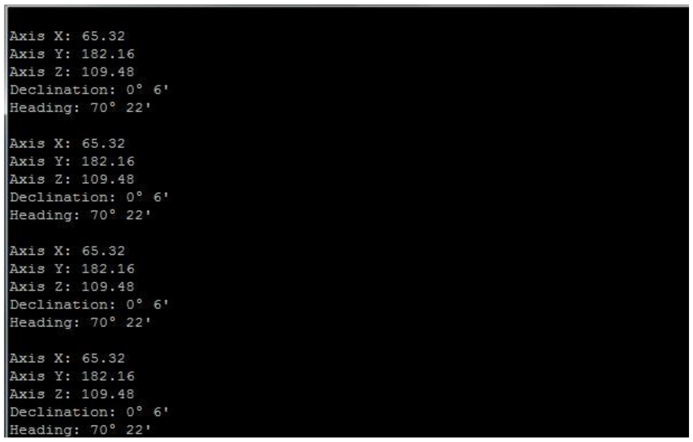


Figure 28: The controlled environment made from cardboard and duct tape

Compass Sensor

The way to test the compass sensor is by pointing the positive x-axis side of the sensor and see where the the sensor's heading value. Comparing it to the real compass, we can see if the heading is 0 if we point the compass sensor north. Then, we can point the sensor to another value and see if the values change consistently. Below is the example of the heading results that we get from the Raspberry Pi terminal when it is pointing north east:



```
Axis X: 65.32
Axis Y: 182.16
Axis Z: 109.48
Declination: 0° 6'
Heading: 70° 22'

Axis X: 65.32
Axis Y: 182.16
Axis Z: 109.48
Declination: 0° 6'
Heading: 70° 22'

Axis X: 65.32
Axis Y: 182.16
Axis Z: 109.48
Declination: 0° 6'
Heading: 70° 22'

Axis X: 65.32
Axis Y: 182.16
Axis Z: 109.48
Declination: 0° 6'
Heading: 70° 22'
```

Figure 29: The heading values from the magnetometer

ERROR ANALYSIS

Some of the many minor complications that were faced when creating this project and how they were overcome will be discussed in this section. Since this system is mobile, power management played a big key in the system design and functionality. The first battery used to power the system did not provide enough current to keep both the MSP and Raspberry Pi powered at the same time, so a new rechargeable battery was implemented in the system, one that provided a larger current. The other problem came from the wheel servos needing a large amount of power as well. This source could not run through the MSP432 board and was instead powered by a separate battery pack.

Another major complication was environmental factors. With the object recognition software to work very accurately, a simple and plain background is required to remove background noise. To remedy this we made a predefined environment that was monocolor except for the intended objects to be collected. The other environmental factor was the wheels having enough traction on the ground. The floor in the lab was found to be too dirty and dusty, reducing the traction between the ground and the robot's wheels. Again this was fixed by building a predefined environment, one that was clean, flat, and provided traction for the tires to work properly. Finally the other issue we faced was the compatibility of software. The code run on the Raspberry Pi for OpenCV used Python 2.7, while the code for the connected compass ran on Python 3. This create errors when trying to run both codes at the same time. This was fixed by running Python 3 code with a function that allowed the Python 2.7 code to be run as well. Along with this, the libraries used for OpenCV had similar names to other libraries already included in the Raspberry Pi software. This was fixed by defining the libraries need through their specific file path, and making sure the code produced the desired output.

BILL OF MATERIALS

Table 2: Components for the object collector device

Parts	**Part Name**	**Manufacturer**	**Price**	**Amount**	**Total**
Secondary Controller	Raspberry Pi	Raspberry Pi Foundation	$40.00	1	$40.00
Controller	MSP-EXP432P401R	Texas Instrument	$13.00	1	$13.00
Camera	Raspberry Pi Camera Module	Raspberry Pi Foundation	$20.00	1	$20.00
Display	OLED	Diymal	$14.99	1	$14.99
Distance Sensor	Analog Distance Sensor GP2Y0A51SK0F	Sharp	$5.16	7	$36.12
Bluetooth Module	Bluetooth HID	Roving Networks	$3.96	1	$3.96
Robot Chassis	Damping balance Tank Robot Chassis shock absorption Platform DIY	Sinoning Robot	$25.00	1	$25.00
Compass Sensor	Accelerometer Gyroscope GY-85		$14.99	1	$14.99
Trapping Mechanism	Clear Plastic Sheet		$3.50	1	$3.50
Power Source	Battery Pack		$49.00	1	$49.00

Servo Motor	MG90s	Tower Pro	$2.50	2	$5.00

Final Cost of the System
This project costs approximately $225.56

FULL SCHEDULE

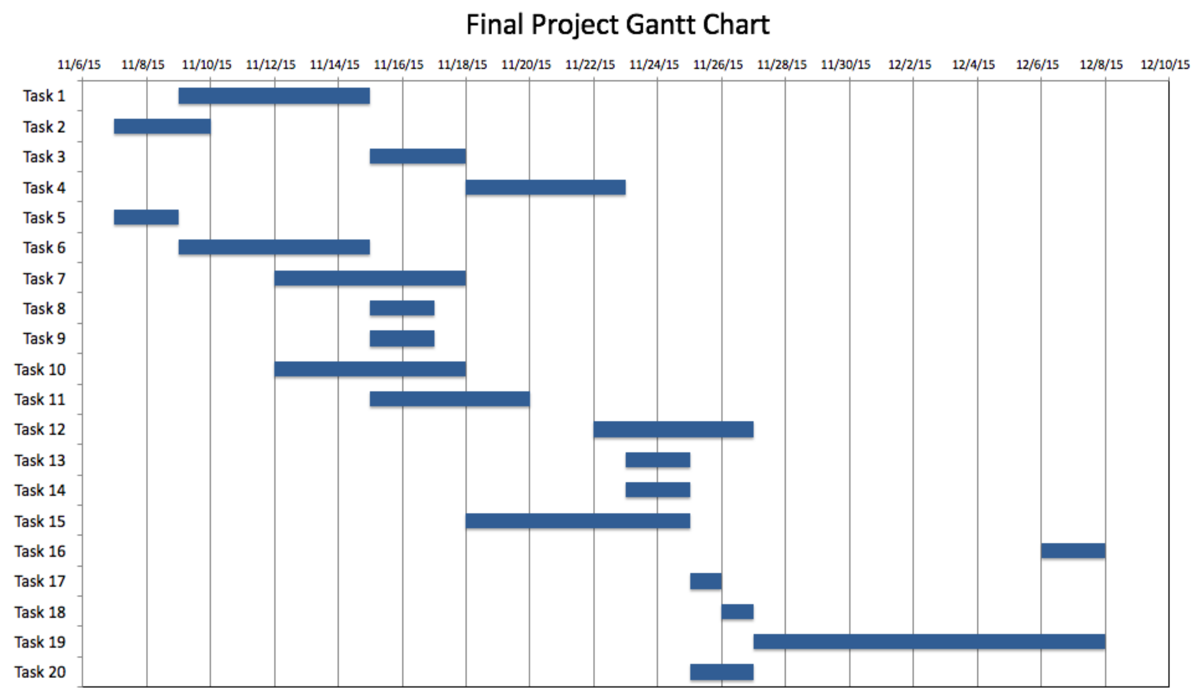


Figure 30: Schedule presented in Gantt chart

Table 3: Detailed task description for final project

Task #	Task Description
1	System Requirement Specification and Schedule
2	Design Specification
3	Configure Raspberry pi camera module
4	Familiarize ourselves with Raspberry pi's peripherals such as GPIO, protocols, and Linux OS
5	Change the format and features of the camera (exchange intensity and pixel size)
6	Analyze the picture coming from the module for the robot to make decisions

7	Familiarize ourselves with MSP430 peripherals such as GPIOs, DMA, ADC, interrupt, and communication protocols
8	Configure the microcontroller to the host PC
9	Data logging from distance sensor
10	Implement IR sensors interface with MSP430
11	Implement H-bridge interface with MSP430
12	Analyze data from IR sensors to determine directions using H-bridge
13	Implement interface of compass sensor to keep track where the robot is currently located
14	Program movement functionality of the robot (Go straight, turn left or right)
15	Design an algorithm to distinguish objects from wall
16	Design and implement the grasping mechanism (trapping)
17	Build the robot and add two servos onto it to control the scoop to trap the object momentarily
18	Configure the user interface of the system (GUI) and OLED
19	Mount board with peripheral devices
20	Mount board to the robot
21	Test object avoidance (boundary testing) to test the accuracy of the distance sensor
22	Build the actual testing environment

SUMMARY AND CONCLUSION

The main goal of this lab is to produce a functioning robot that is able to identify a piece of garbage, pick it up, and place it in a designated location or recycling bin. It has been tested for basic functionality and meets the specifications and requirements. If there was more time to the class and funding was not an issue, changing the mechanical parts of the robot would be an easy way to improve its functionality. Buying a better body, with newer tires and more space to add sensors would have made our project run much more smoothly. If we had time to add our libraries to OpenCV, that would have added to the complexity of the project. To make it more user friendly, being able to add app functionality or a web service would have been extremely useful to implement.

The autonomous recycling robot was built and tested using many of the skills learned through numerous EE classes. In conclusion, this project taught us how to properly use the MSP432 and the Raspberry Pi along with peripherals such as Bluetooth, compass sensors, distance sensors, the Raspberry Pi camera module, and servo motors which can be useful for further projects and outside of a classroom environment.

APPENDICES

Final version of software sources

- compass.py is used for running the image capturing/ processing code and compass sensors that switch between Python 3 and Python 2 back and forth
- msp_rasp_1.py is used for the actual object recognition as well as sending flags through UART to MSP432
- main.c is used for running the logic used on the MSP432 board
- enviro_gui.m is used for the GUI of the entire system