BroBot

Hey Bro, can you watch my stuff?

Jacob Stewart

Richard Landau

Sarah Patten

Anson Contreras

Table of Contents

1.0 Executive Summary 1

2.0 Project Description: 1

2.1 Project Motivation and Goals: 1

2.2 Objectives 2

2.3 Security Detail 6

2.3.1 Item Identification 6

2.3.2 Camera 7

2.3.3 Microcontroller 8

2.3.4 Wireless 10

2.4 Navigation 10

2.4.1 Chassis 10

2.4.2 Hardware Considerations 11

2.4.3 Software for Movement 13

2.5 App 14

2.5.1 Design/Flowcharts 14

2.6 Power Considerations 15

2.6.1 Battery 15

2.6.2 Voltage Regulators 15

3.0 Research Related to Project Definition 16

3.1 Similar Projects and Ideas 16

3.1.1 B.R.A.V.O 16

3.1.2 Knight Sweeper 4200 17

3.1.3 R.C Ghost Rider 18

3.1.4 Track Detector 19

3.2 Relevant Technologies 20

3.2.1 Wi-Fi 20

3.2.2 Roomba 20

3.3 ARM Microcontrollers: 21

3.3.1 Texas Instruments Tiva C Series and Other Considerations: 21

3.3.2 STM32F407VGT6: 24
3.3.3 ATSAM4S16B 26
3.3.4 Conclusion of ARM Processors: 27
3.4 Low Power Microcontroller 28
3.4.1 MSP430 28
3.4.2 Arduino Uno 29
3.4.3 PIC 29
3.4.4 Conclusion for Low Power Microcontroller 30
3.5 Movement 30
3.6 Chassis 31
3.6.1 4WD Robot Chassis 32
3.6.2 Aluminum 4WD Robot Chassis 33
3.6.3 Baron-4WD Mobile Platform 34
3.6.4 Pirate-4WD Mobile Platform 34
3.6.5 Dagu Rover 5 Chassis 2WD 36
3.6.6 Conclusion of Chassis 37
3.7 Navigation 37
3.7.1 Algorithm 37
3.8 Sensors for Navigation 40
3.8.1 Sharp GP2Y0A02YK0F 40
3.8.2 Sharp GP2D120XJ00F 40
3.8.3 Pololu 38 kHz IR Proximity Sensor 41
3.9 Camera 41
3.9.1 Camera Setup 41
3.9.2 JPEG Image/Video Compression: 43
3.9.3 IR Motion Sensor 45
3.9.4 TTL Serial JPEG Camera: 46
3.9.5 MT9D111: 47
3.9.6 Conclusion for cameras: 49
3.10 External Memory 49
3.11 Android Application 52
3.11.1 Communication 52
3.11.2 APIs 53

3.11.3 App Picture Manipulation 54
3.12 Voltage Regulators 55
3.12.1 LT1121CN8-3.3 55
3.12.2 LT1587CT-3.3 55
3.12.3 LM2594N-3.3 56
3.12.4 Conclusion for Linear Regulators 56
3.13 Bluetooth Modules 56
3.14 Batteries 58
3.14.1 Lithium-Ion 58
3.14.2 Nickel Metal Hydride 59
3.14.3 Nickel Cadmium 60
3.14.4 Lithium Ion Polymer 61
3.14.5 Battery Conclusion 61
4.0 Project Hardware and Software Design Details 62
4.1 Initial Design Architectures and Related Diagrams 62
4.2 Item Watcher 63
4.2.1 Hardware Configuration 63
4.2.2 Camera 64
4.2.3 Item Watching Program 70
4.2.4 ARM Microcontroller 72
4.3 Navigation 80
4.3.1 MSP430G225 80
4.3.2 Interfacing with the IR sensors 83
4.3.3 Algorithm and Interrupts 84
4.3.4 Communication with the ARM processor 87
4.4 Wireless System 87
4.5 Pololu 38 kHz IR proximity Sensor 89
4.6 Android Application 91
4.6.1 Programming Language 91
4.6.2 IDE 91
4.6.3 Libraries and Tools 92
4.6.4 Compatibility 92
4.6.5 Communication with hardware 92

4.6.6 Permissions 93
4.7 Bluetooth module 93
4.8 Power Protection 95
5.0 Design Summary of Hardware and Software 96
5.1 Item Watcher Subsystem 96
5.1.1 Hardware Configuration 96
5.2 Navigation 97
5.2.1 Hardware Configuration 97
5.2.2 Steering Mechanics 98
5.2.3 Algorithms and Interrupts 98
5.3 Android Application 98
6.0 Project Prototype Construction and Coding 99
6.1 Navigation 99
6.1.1 Sensors 99
6.1.2 Movement 100
6.2 App Integration 101
6.3 Camera 101
6.3.1 Communication with Camera module 103
6.3.2 Data extraction by ARM Processor 104
6.4 PCB 105
6.5 Integrating Vision Software 106
7.0 Project Prototype Testing 107
7.1 Item Watcher Subsystem Hardware 107
7.1.1 Camera communication and data flow 107
7.1.2 Communication between subsystems 108
7.1.3 Test LEDs 109
7.2 App Stand Alone Testing 109
7.3 Image Tracking Testing 110
7.4 Navigation Testing 110
7.4.1 Sensors 110
7.4.2 Software for Movement 111
7.5 Prototype Testing 112
7.6 Battery Life Testing 113

7.7 IR Sensor Testing 114
8.0 Administrative Content 115
8.1 Administrative Content Management 115
8.2 Administrative Content Milestone 116
8.3 Budget 120
Appendix A 122
Appendix B 124
Appendix C 125

1.0 Executive Summary

The future of personal security is here. As alarm technology improves, homes are becoming more and more secure from intruders looking to rob you of your hard-earned belongings. However, there is one place where members of society are more vulnerable than ever – the library, a place where people go when they do not want to be disturbed and need to get some work done.

In the library, all your things are typically stacked up into one small area. This is fine when you're studying, but if you have to step away to go to the bathroom or make a phone call, it becomes a potential jackpot for thieves. This is where BroBot comes in. Using computer vision technology, BroBot is a mobile robot that seeks you out when it is called, and has your back by watching your things when you need to step away. Housed on a mobile chassis, BroBot consists of a camera feeding images into an algorithm hosted on an STM ARM processor. This processor maintains a Bluetooth connection to the user's smartphone, where a custom Android application allows the user to monitor their items in real-time and be notified immediately if something gets stolen.

The ARM processor also connects to a low-powered MSP430, housing our motion algorithm. Given the user's approximate location within the library, the algorithm steers BroBot in the correct direction, avoiding both people and immobile obstacles while he traverses the labyrinth. The motors to control BroBot are a part of the mobile chassis, allowing movement.

BroBot is powered by a battery, allowing it a full range of indoor motion with extended life. This battery will be rechargeable, allowing BroBot to be charged at night after a long day of item rescuing to be ready again the next day. Using a rechargeable battery will dramatically cut down on the cost of long-term use.

BroBot attempts to solve a real world problem by implementing several different areas of electrical and computer engineering. Power and circuit analysis, computer vision, embedded hardware, Android programming, Bluetooth, and motion/object avoidance software all come together to achieve the goals of BroBot.

2.0 Project Description:

2.1 Project Motivation and Goals:

Across the country, studious students gather in libraries to prepare for exams, work on projects, and finish homework. While some students prefer to work with a group, there are others that can only be productive alone. In engineering these

reclusive studiers seem to be more prevalent, as observed from within. When these students are off trying to be productive, they come to face a dilemma when they're in need of a break. Should they pack up their study materials, take them along, and risk losing their prime study location? Or, should they leave their valuables unattended and risk having them stolen? This situation is faced regularly, and needs a viable solution.

BroBot is the proposed solution for this problem. This robot will watch the user's belongings when they need a break. Upon request, a student in the library will request BroBot from the library with a mobile app. If available, BroBot will travel to the location the user gives by the use of infrared sensors and a general map of the library. This map will mainly be used just as a general direction that BroBot needs to travel in. Upon arrival, BroBot will have an extending camera that will peer over the edge of the table to have sight of the objects. The user will select which objects need to be watched and then activate him. The user can then leave their belongings knowing BroBot is on guard. When watching the user's valuables, BroBot should be able to make sure none of the user-selected items disappear. If by chance they do, BroBot will sound an alarm, take a picture of the thief, and send a text to the user communicating the situation.

To be able to complete all of these tasks, BroBot will have a Bluetooth to connection to the phone application. This will be used to initially request BroBot to come to the user's table, allow the user to select which items need to be watched, as well as obtaining a wireless number for SMS notifications and alerts. There will be infrared sensors that will be used in navigation to the user to avoid collisions with people and other objects along the way. There will be a camera that will be used in the detection of the actual objects. The images taken will be processed to determine the amount of change within the frames, and whether an object completely disappears. BroBot will also be able to use the object detection algorithm to also be able to tell the difference between the user's objects and another person's things. An alarm will be implemented to alert people in the area that something has been taken, as well as to scare the thief.

In the future, we would like BroBot to be used in multiple college libraries. The library could have multiple BroBot's to allow multiple users at once. Another goal would be to create a portable version to allow the user to have a personal object detector. This portable version wouldn't need to travel, so he would simply watch items and sound the various alerts.

2.2 Objectives

To accomplish the goals of BroBot, many subsystems need to work in harmony. Without any one of these subsystems, BroBot would not function and the defenseless items left to its care would be completely vulnerable to poachers nearby. In this section we will discuss each of the subsystems as a whole.

Further into this document, we will discuss each of the subsystems and their workings in detail.

Figure 2.2-1 gives an overview of each of the subsystems of BroBot. This chart divides BroBot's hardware systems from the software systems. The hardware components consist of the chassis + motors, power system, and the physical integration of the microcontrollers. The software includes both the algorithms used in BroBot and the software to communicate between each subsystem. Each of these software subsystems must be coded to both do what they are designed to do and also must establish and use their connection with other subsystems.

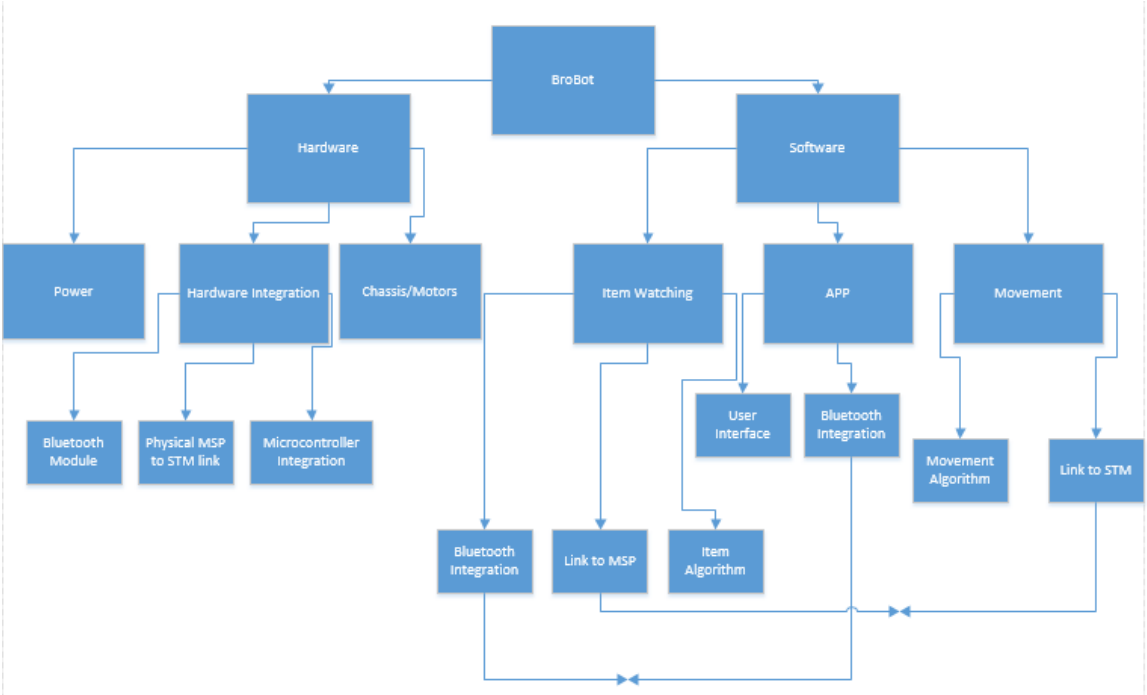


Figure 2.2-1 created by Richard

The first and arguably most important subsystem is the power system. All of the other components, from the smallest of users, the camera, to the largest, the motors, require electricity to function. Without this power, all the other subsystems are worthless. Although more economical forms of power were considered, they all have drawbacks that make them unusable for BroBot – for example, solar power cannot be used in the indoor setting of a library. We will be going with what has been the staple of portable power use for decades – the battery.

The core of BroBot revolves around the item watching subsystem. This system physically consists of a camera connected to a high-power ARM processor. This processor will be running BroBot's item watching software. The camera will continuously feed pictures it takes to the software. The software will decide if something about the picture has "changed," and if so, will set off BroBot's alarm. If the picture is similar to the original one, it will continue to stand by and watch,

constantly taking new pictures to compare. A Bluetooth module is included on the processor to allow it to constantly communicate with the user interface on the app.

This processor will be connected to a second microcontroller, this one a low power MSP430. This connection will serve to pass the destination information from the ARM processor to the MSP430. The MSP430 will be in charge of the software dictating BroBot's movement. This simple navigation software will tell BroBot to head in the direction of his destination. IR sensors will send a signal to this processor if something is in BroBot's way. In this case, the software will wait a few seconds to see if it is something that is going to move out of the way. If not, it will turn and attempt to find another route.

All of these systems are on board a portable chassis, equipped with a motor and wheels. This chassis will be the body of BroBot, allowing it autonomous movement. It also provides the base upon which the camera is mounted, allowing BroBot to have a downward-angled perspective of the items it is monitoring. This is important to eliminate false alarms due to background movement.

The user interface to BroBot will take the form of an Android application, usable on most Android smartphones and tablets. This app will allow the user to select a destination for BroBot to travel, and will receive pictures of BroBot's field of vision to ensure security of watched items. The app will communicate with the item watching software via the Bluetooth functionality of the device. If something is stolen and the app is in range, it will notify the user and will be sent updated pictures. If it was a false alarm, the user can choose to send BroBot back into watch mode.

Figure 2.2-2 gives a visual representation of the information flow through BroBot's subsystems. The user receives their information and is able to send commands through the Android application on his or her phone. This information is shared only with the STM processor, which contains a Bluetooth module enabling it to send and receive data via the medium. This processor has the item watching software loaded on it, and communicates both ways with this software. The software also takes in inputs from BroBot's camera and uses it in the analysis.

The STM processor is connected with the MSP430 processor. The main purpose of this connection is to allow BroBot's destination information, received from the user's choice in the Android application, to flow from the item watching software and get passed to the MSP430 where it will be used. The MSP430 runs the movement software, and uses the destination to decide where it will go. The resulting decision is passed to the motors on the chassis, which turn the wheels and enable BroBot's movement. Facilitating all this is the power system. Although

this power system does not require an exchange of information with any subsystem, it will connect to each one to supply the power required.

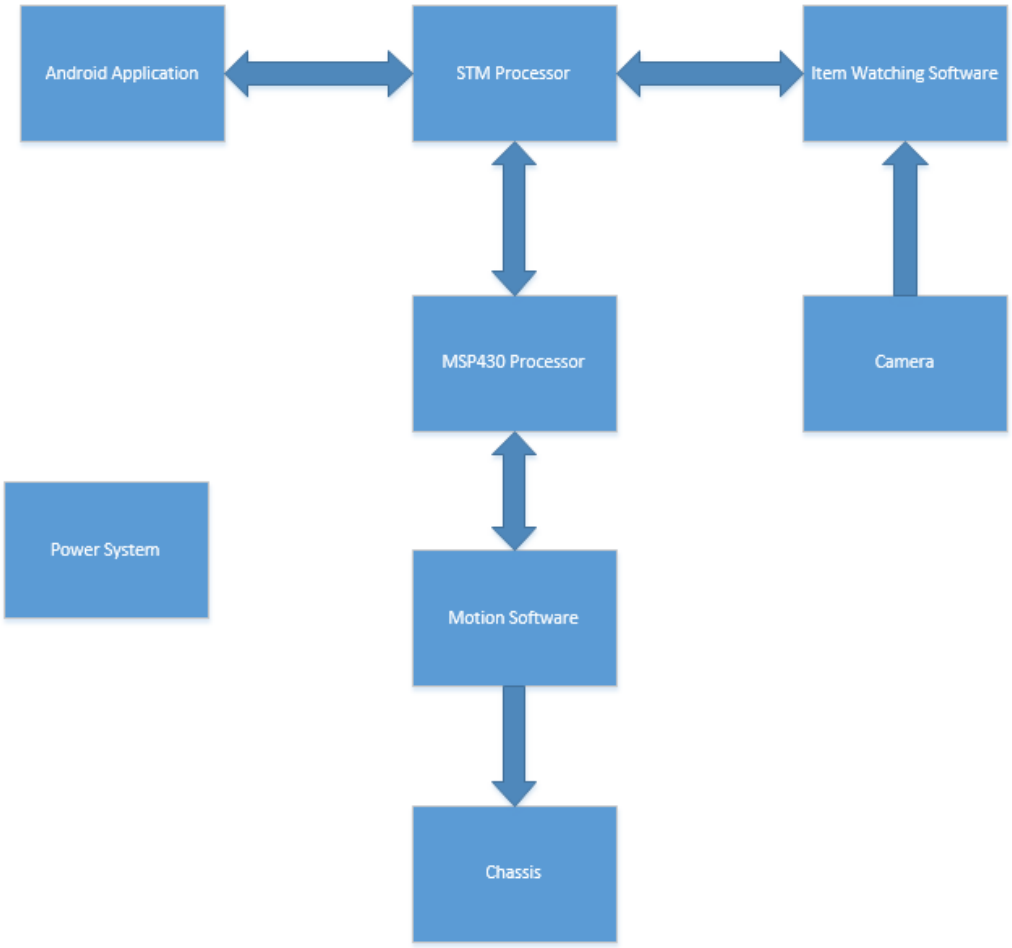


Figure 2.2-2

Figure 2.2-3 shows the interaction between the software and the external information they involve. The Android app shares information back and forth with the item watching software. The item watching software takes as inputs this information and the jpegs inputted from the camera. The item watching software also receives the information about BroBot's destination, and shares this with the movement software. The movement software sends the signals to the motors built into the chassis.

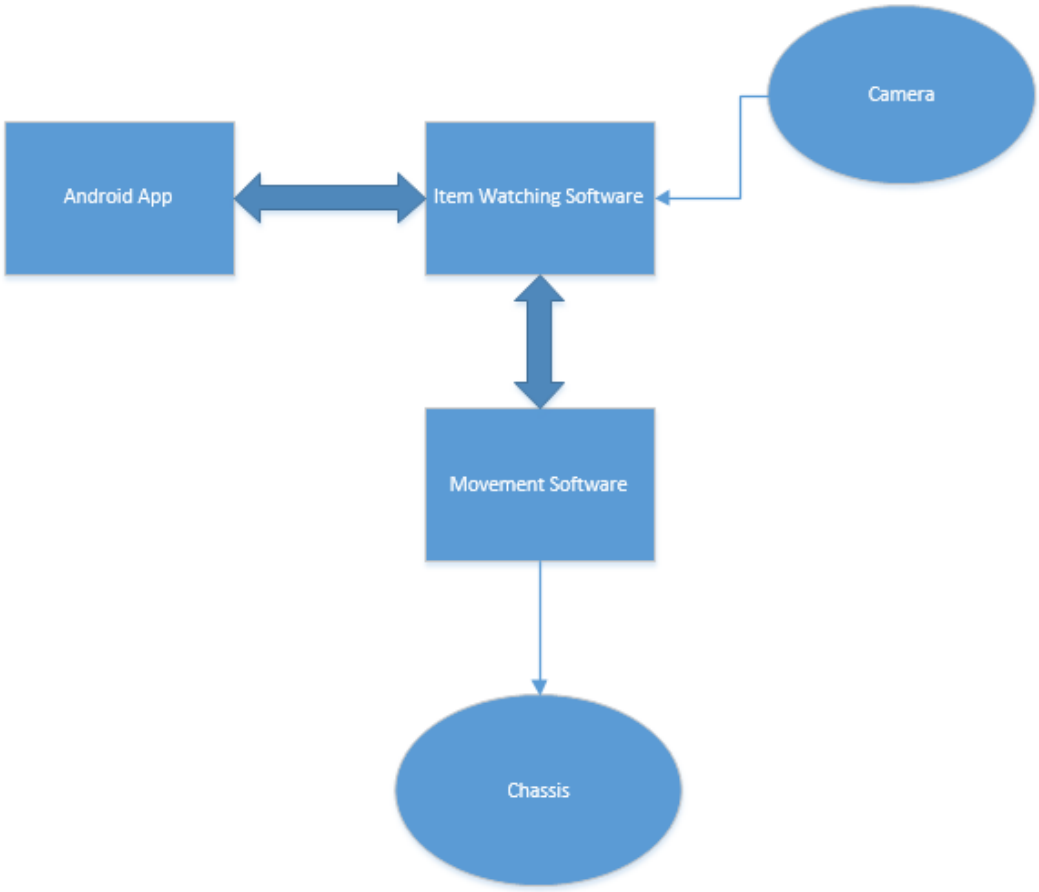


Figure 2.2-3

2.3 Security Detail

2.3.1 Item Identification

The core of this project relies on watching the user's items and knowing when something is stolen. Unfortunately, computers do not have very reliable ways of knowing what something is by looking at it. Because pictures are just stored as a matrix of numbers, where these numbers represent the color and brightness of the picture, it is nearly impossible for a computer to be able to distinguish between two separate objects or tell the difference between two pictures.

Because of these limitations, we are forced to use a more rudimentary technique. We will take a picture of the area when BroBot is activated. This picture will serve as a base. Every few seconds, a new picture will be taken and the difference of the two will be taken using matrix subtraction. After this, the magnitude of this difference will be calculated. If this difference differs from a threshold value, it will

trigger the alarm, if it is not, it will not. This threshold should be a value where small changes like brightness does not set off the alarm, but a big change like something disappearing will. Because slow changes such as an encroaching shadow or change of brightness will eventually cause the alarm to be falsely triggered, we will occasionally update the initial picture in an effort to trigger the alarm for only sudden changes, and let slow changes get absorbed.

2.3.2 Camera

One of the main features of BroBot will be its item watching subsystem. To fully implement this a camera while be needed on the robot itself. This camera will be interfaced with a microcontroller which will take the image and store it for future computations. The camera will be small enough so that it will not impede on BroBot's ability to navigate through the given terrain. Also the camera will be on a perch above the body of the robot, which is another reason for the camera to be small and light weight so the arm that holds it won't need to be too heavy. The camera will only be used during the image watching process it will supply, when asked from a microcontroller, a picture will be sent from the camera to the microcontroller.

The picture that will be sent might not need to be compressed if there is enough processing power and memory in the microprocessor to manipulate larger picture files. But since the communication between the processor and the camera might be serial it would be much faster if there was some type of image compression done by the camera itself. While serial communication isn’t necessary some type of image compression would be ideal for this project. The compression from the camera would be most useful if there was a way to control the quality.

Since it is hard to know the quality of the taken picture we would it’s hard to know if the image is of a good enough quality to do the proper manipulations of the image. With an adjustable quality comes the ability also to scale the byte size of the image, which can decrease the amount of time the microcontroller will need to process the image. Though this scaling will not be linear we can still obtain really nice compression through the lower qualities of images.

The images themselves will need to be of a high enough quality that we can discern different items in the image. Therefore when choosing a camera with the ability to scale its quality it will be extremely useful in the debugging phase of our building to be able to change this setting. Another way the camera can easily scale the size of the image is through adjusting the resolution of the image coming in. The ability to change the resolution would be one of the easiest ways to adjust the overall size of the image.

Another item for consideration is that the camera must be able to send a picture two times a second, which means that the serial communication must be fast enough to compensate for the size of the files that will be sent. The data that is

sent from the camera can also come from a parallel data stream as long as the microcontroller that is picked can interface with it. Data streams in parallel are much faster than a serial wire but also take up a lot more pins on the microcontroller. Also the microcontroller that is picked will have to be a lot more powerful to take in that amount of data easily.

Since BroBot will run on battery power during the main operation it will be ideal to get a low power camera so the battery will have enough power left to be able to get back to its home station. Also the camera will need to be able to go into a lower power state, though this can also be controlled by a microcontroller that will power up the camera when it needs to be used.

The ability to zoom would be a beneficial functionality of the camera. With the ability to zoom comes the ability to limit the amount of items or objects that can cause a problem with BroBot's watching program. But with a zoom feature the camera will need to be able to auto focus from the zoom feature. Another nice but not needed feature would be being able to grab a video feed from BroBot's camera. Video would take a much more powerful processor to get what we want out of it, also it would take a lot more memory then photos would. A zoom feature might be nice if the user only wants a small part of the field of view watched. But since the user is choosing what items the processor will watch this feature isn't a necessity.

Instead of using a full implemented camera a CMOS sensor matrix could be implemented to retrieve a picture. While this would give total control of the raw data coming it this would be the most difficult idea to implement. Also we would have to compress the images ourselves. This could be done using a specific DSP that was designed for this type of functionality.

2.3.3 Microcontroller

For the item watching process a microcontroller is needed to communicate with the camera and also to manipulate the image and tell the user if something is wrong. Also the microcontroller will be able to take an input from the user wirelessly to turn on the camera and send a picture back to the application so the user can define what items he/she wants BroBot to watch. Along with communication with the user the microcontroller will need to be able to either interface with SRAM or external ROM and will be able to store multiple pictures on the processor itself.

The external memory that could be interfaced with the microcontroller will need to be large enough to hold a good amount of pictures, around a GB would be more than enough for pictures. This external memory also needs to be able to send information fast to the microcontroller. This is to make sure that the memory will not slow down the image calculation process. Also the external memory will need to interface well with the microcontroller.

Since the microcontroller will be need to perform image processing duties it will need to be quite fast and have the ability to do floating point operations. Though a DSP would be work great for the image processing on this project, it would increase the amount of money that we would need to spend. Since a good amount of powerful microcontrollers have cheap development boards they would be a better choice. A DSP development board is very expensive and also has a high learning curve when it comes to using it properly. The program for image processing and to communicate with the user will be quite large so around 1 MB of ROM will be needed. As stated early the processor will need to be able to store at least 1 MB of data pertaining to the images that the camera is sending them.

The microcontroller will also have some type of serial communication since it needs to communicate with multiple items including the camera, blue tooth module, and other microcontrollers that control the movement of BroBot. These communications are vital to the overall success of BroBot. The microcontroller will also be able to go into low power mode when not in operation, so to conserve battery power. An arm processor will be strong enough to do the calculations but also won't take a lot of power and will be much cheaper to test then a dedicated DSP built for image processing.

A small goal for BroBot is the ability to check older images or be able to pull back older images so the user can look at them. If the system takes a picture every 2 seconds and BroBot will watch the item for 15 minutes that will be 450 pictures to store. While this is a large amount of data to store on the microcontroller external memory can help with this a lot. If the pictures take 10KB to store that would be 4.5 MB to store. This feature will act as a way to look back at when the items were taken and also for the user to help catch the perpetrator.

An interesting problem is the amount of money that we are willing to spend on a microcontroller. Since this is just a prototype with no real intention of going into production a microcontroller with a cheap development board is needed. Some development boards are wonderful pieces of testing technology, though that is not needed for this project. For example some ARM microcontroller development boards come with a LCD screen already built in along with a capacitive touch screen. While these features would be really interesting to work with they are not needed for our project and would be better if there was a development board that is much cheaper with less extra peripheral.

Another consideration is a second low power microcontroller that will control the data flowing from the camera to the processor that will be doing the calculations on the images. This will certainly increase the amount of power the main processor can use towards image manipulation and calculations. A problem with this is it will complicate the system a little more, but the microcontroller can also have control of all data going in to the main processor, and tell vital information

pertaining to the data coming in or out of the more powerful microcontroller. In the system the low power microcontroller will act as the master while the more powerful controller will be the slave. Even though it will oversee the entire data flow of the system the microcontroller can be much less powerful then the image processor, this is because the microcontroller wouldn't do many calculations when addressing the camera and the image processor.

2.3.4 Wireless

To communicate with the user BroBot will use a wireless protocol that will be connected to the main microcontroller that is doing the image processing. The protocol will only need to work in a short range (<10 m) since this particular communication will only happen when the robot is near the user. The wireless system that will be used to implement this protocol will need to have a low power option so that the wireless module that is used will not drain the primary battery on BroBot. The system that is implemented will also need to be small enough to fit inside the robot.

Communicated wirelessly will be the first image for the user to pick their items and also to switch BroBot into item watching mode. Therefore the method that will be used only needs to send one stream of information to the microcontroller. Also this transferring of information needs to be accessible for Android applications since control of BroBot by the user is through an Android application. Since BroBot will be used in the library setting then it would be ideal to use a wireless system that can work well with many people using wireless products. Also this system might be able to integrate into the overall wireless system that is being implemented for communication to the authorities when an item is stolen.

2.4 Navigation

2.4.1 Chassis

As we shall be adding things on top of our foundation, the base should be sufficiently strong. Along with the chassis being strong, it should be able to support all of the necessities of the project, which include the processor, the microcontroller, the various sensors, as well as the camera and its support fixtures, and any additional items that could be added. The camera will possibly be added on to a tube to replicate a telescope to peer over the edge of the table. All these add-ons will be fairly light, but the base should be able to support up to five pounds, just in case. This means it has to be made of a sturdy material.

The chassis should have adequate steering capabilities. There will be times in the course where BroBot should be able to handle ninety degree turns, as well as navigating around still standing objects. Some things that will affect the ability to turn will be the type of steering as well as the wheels.

The final consideration for chassis selection will be the size. For our project there will need to be enough room to build on top of it. Possible considerations for size would be between eight and fourteen inches for both length and width. This is so that we can add all the necessities for the project, as well as possible additions that could be created later.

2.4.2 Hardware Considerations

The hardware for the navigation system of BroBot needs to meet a couple of simple parameters. One of these parameters is the ability to control at least 2 motors at one time, which will be used for the overall movement of the robot and the steering the robot will need to perform. This can be done a couple of different ways, with a microcontroller that has built in motor drivers, or external motor drivers that are controlled using a small low power MCU. A microcontroller with a motor driver might be too much for our project, though if a good compromise can be found then that could work. If we use separate motor drivers then a much easier to work with microcontroller can be used. Another big consideration for the navigation system of BroBot will be the microcontroller that will be the brains of the operation

2.4.2.1 Microcontroller

The microcontroller that will be used for the navigation will need to be low power and will not need to do too many calculations. Since all it is doing is pulling up a location and then instructing the motors where to go. Also the microcontroller will need to be able to communicate with the processor that is in charge of the entire project. There are a couple of good serial communications that could be used, I2C, SPI, and/or UART. Depending on the amount of pins that will be used we will choose the corresponding protocol. The microcontroller will need to be able to hold the locations of the different sections that it will go to. Since the number of locations will be small the ROM of the microcontroller will not need to be too big. Also since the microcontroller will not be doing a lot of different mathematical operations and will not need to store too many different variables then the RAM doesn't need to be too large.

The microcontroller will not need to do any floating point operations, which will greatly reduce the cost and the power consumption of the overall system. This will also mean that processor will not need to be too large, which is great since it will be on our robot while the robot is moving, the less weight the less amount of power that will be needed to move the robot. A good amount of GPIOs will also be helpful when interfacing with different sensors that the robot will need during the navigation period.

There is a lot of flexibility in the choice of microcontroller since it isn't doing anything extremely power intensive, and because of this we can choose a

microcontroller that can be easily tested, or even we could use a specific family of microcontrollers that might be able to do the job at hand. Another consideration that must be looked at is the overall price of using the microcontroller.

The navigation system is only a part of the overall robot that will be implemented. Therefore the microcontroller will need to be able to go into an extremely low power mode when the system isn't in use. The ability to wake up from these different modes will also be extremely useful in the system, since it communicates with the main processor via a serial line. It would also be great if the microcontroller could do all that is needed in a low power mode, since a lot of power will be used when this subsystem is working.

Since we will need a development board to be able to test the microcontroller, a cheap microcontroller with an expensive development board should not be the answer. Therefore the microcontroller's development board will need to be inexpensive and straightforward to use. There are many great low power microcontrollers that have inexpensive development boards.

2.4.2.2 Sensors

The sensors are what the robot will use to be able to see what is going on with the real world. The sensors will need to also be able to easily interface with a low power microcontroller. Which means that it will need to take up a small amount of pins and also not use too much power. The power consideration is very important since it will be used when the overall system will be using the most amount of power, during the movement of the robot. Also to be able to interface easily with the low power microcontroller, it would be advantageous if the output of the sensor is a digital output.

A problem that can occur with a digital output is the lack of calibration that can occur. Since only a one or a zero is outputted it is hard to adjust when the sensor will see something, and this could prove to be problematic during the navigation coding if the sensor says something is in the way when in actuality nothing is in the way. But with analog output the microcontroller will have to convert that output so that it can give data that can be used. Though to get around that we could use external Analog to Digital converters. This would further complicate the system that we want to implement, but might be the only way for the system to get a good dependable reading from the sensors.

The sensors will also need be to light weight, since more weight will mean more power from the system overall during movement. But they also need to be able to see at least 10-30 cm ahead of themselves, so that when they do something the robot can come to a stop easily and not run into whatever it sees. Since the sensors will be used in a library setting they will need to be able to sense with a lot of ambient sound and a lot of other electrical devices being in use.

2.4.2.3 Motors

The motors will need to be able to pull the amount of weight on BroBot and more. They need to do this while not using too much power on the system, since after the navigation system is used BroBot will then need to be able to watch the items and then finally come back, so power consumption is a large part of the overall system. The ability to control these motors digitally is also a consideration.

Just because a motor can pull a lot of weight with lower power doesn't mean that it should be used in our project. Control is extremely important since we aren't using anything in the navigation that will tell us our location. This means that we will need to keep a tab on where we are. This can be done if you know how many rotations the motor has made and how big the tires are. With that information it is easy to see where the robot is. Any easy way to accomplish this goal is with stepper motors. While stepper motors give a user a lot of control they are much harder to implement and interface with a microcontroller, even with a stepper motor driver.

2.4.3 Software for Movement

The software in charge of vehicular transportation will have a few major components. Starting, there will be a general route that should be used as a guide for BroBot. What this would entail is finding a way to give BroBot a sense of direction. He would have a starting and ending location given to him, and a preprogrammed layout of the library in which he'd be operating. This would not serve as the only component of the movement software needed however.

Another aspect of the software will be object detection. As he will be in a library, there will be people walking around, as well as tables and chairs that can be moved. He will need to be able to detect any object and determine if he should wait for it to move, if it is a person, or reroute because the object is inanimate and won't move.

Combining these two requirements, our BroBot will be capable of incorporating a movement algorithm which continuously reads in from the sensors, as well as ensures that he is still going the correct direction of the object. While doing this, he should be able to make sharp turns, travel at a relatively fast speed, and stop quickly. For the stopping, the sensors will be used to determine the distance of objects directly in front of it. If there is something there, it will slow to a stop before it can collide with it.

2.5 App

2.5.1 Design/Flowcharts

Figure 2.5.1-1 shows the flow control from the app's perspective. When the app is started, the user is prompted for their location area number. When the BroBot is activated, the app is on standby while BroBot monitors the items in its sight. If it detects an item has been stolen, it sends a timestamp to the user along with a picture of what it sees. If the user is in Bluetooth range, the app sends regular picture updates of its field of vision whether or not the alarm has been triggered. If nothing has been stolen, the user can choose to disable the alarm and allow BroBot to continue monitoring. Every minute or so, the initial picture will be updated to take into account changes of shading or light.

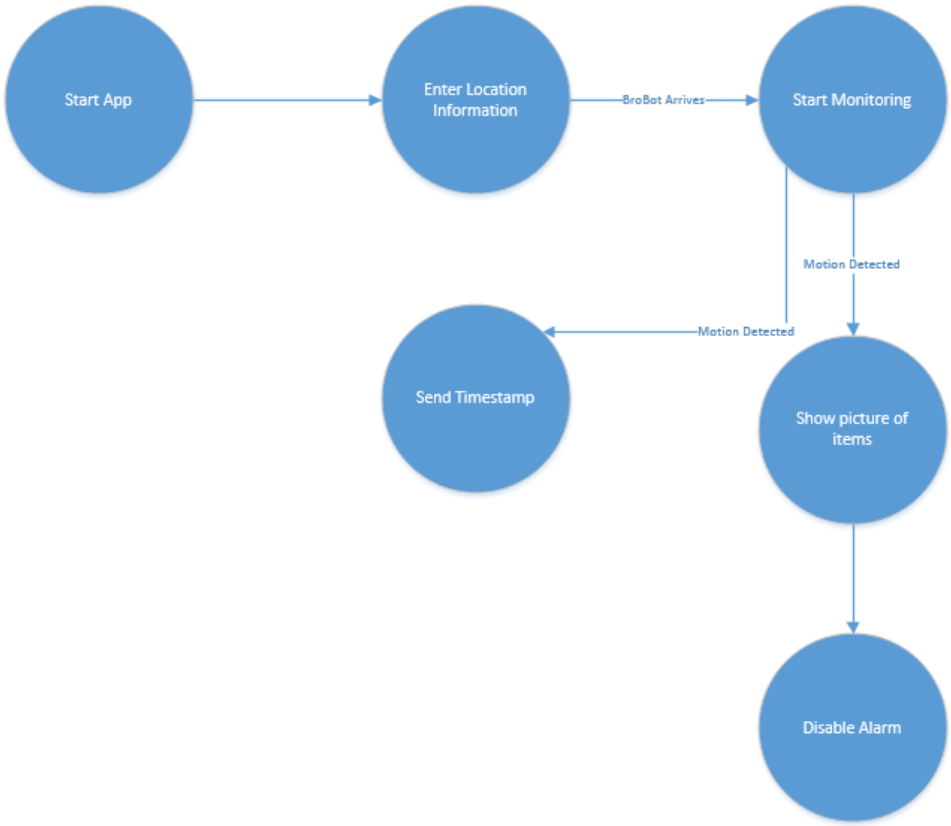


Figure 2.5.1-1

2.6 Power Considerations

2.6.1 Battery

BroBot will be a mobile robot that will need to be wireless, due to this constraint BroBot will have to run on a battery system. The battery will need to be able to supply a constant voltage and current to the system, along with the battery a power grid will need to be implemented in the system, since a lot of the items on the robot have different voltage and current demands.

Since the robot will be used in a library setting and will be used multiple times during the day a rechargeable battery is ideal. Since the robot will go back to its starting location that location can have a charging station for the robot. The weight of the battery will need to also be looked into, since the robot will be using its own power system to move the robot to the user. The heavier the robot is the move power will need to be used by the motors to get the proper movement.

The voltage of the battery will also need to be higher or the same of the highest voltage requirement, this is to ensure that the system will work properly and there will be no problems with the power consumption of the system. It should also be noted that it is much easier to step down DC voltage than it is to step up a DC voltage source.

One simple configuration of the system would be to have all the parts on one battery system. This means that all of the voltage buses will come straight from the battery and would need to be created using some type of voltage regulator to get the needed rails. While this type of system would be easy to implement and regulate there could be other configurations. Since the most power draw in the system will be the motors during the navigation operation a separate set of batteries could be used for that system. If this system were to be implemented then a smaller battery could be used for the electronics of the system. The problem that this would bring up is charging the overall system.

The last consideration for the battery will have to be the price of the battery. Which battery that is chosen can greatly increase the price of the overall system. Since our robot will not be really heavy we can do without an expensive battery. The price does go up with the ability to recharge the battery, which is going to need to be in our minds when choosing our battery system.

2.6.2 Voltage Regulators

Since we plan to only use one voltage source, i.e. a battery, that source will need to be divided into different voltage rails. An easy way to execute this is by using

voltage regulators with a voltage divider circuit. The voltage regulators will need to be somewhat inexpensive and will also need to operate in a normal indoor temperature range. Since the electronics in the robot will not be enclosed in a box the electronics will be able to dissipate heat a little easier than if the robot's electronics where open to the outside.

Also voltage regulators can have a large variation in size. For our project we will use regulators that can easily fit on a PCB board inside the chassis of the robot. The regulators should be lightweight as well, which should limit their size.

With all of the components that we have chosen only one voltage needs to be supplied to the system. This is because the chassis that was chosen already powers the motors for us. This will severely shrink down the power consumption of the control section of the robot. Every part in the circuit board needs or can have 3.3 Volts, therefore we will only need to produce one line of 3.3 Volts. It might be advantageous to add another line at 5 V if we plan on external memory, since a lot of the external memory IC's researched take that voltage to work at the proper speeds.

3.0 Research Related to Project Definition

3.1 Similar Projects and Ideas

To properly research the development of BroBot, some past projects were examined to gain an understanding of some working implementations of different attributes. Using the information obtained, we may consider similar solutions to handle our problems. The projects that seemed worthy of note, due to their similarity to the BroBot, were B.R.A.V.O, Knight Sweeper 4200, RC Ghost Rider, and Track Detector. Each of these projects had some similarity to our vision of what BroBot will be.

3.1.1 B.R.A.V.O

This project was an implementation of a fully autonomous vehicle that can travel between two points on a designated path, avoid collisions, and follow the basic rules of the road. This type of technology is needed as a way to increase the safety in the most dangerous situation a person can put themselves in.

B.R.A.V.O relates to our project because it involves navigation of an RC hobby car, and BroBot will focus on similar navigation of a chassis. Also, this project uses image processing, which we'll also be incorporating in our project. While they have a predetermined path to follow, and specific locations (roads) that they can travel on, the BroBot will still need to travel between two designated points, but there won't be designated locations he can travel, due to regular human

movement through a library, as well as the movability of various objects, mainly tables and chairs, throughout the layout of a library.

After their in depth research of the possible choices for the project, some decisions were made about the parts to use. For the camera, the Link Sprite JPEG color camera was chosen for its price, size, picture size, and usability with available computer vision algorithms. For detection of objects, as well as some line detection, the Maxbotix LV-EZ1 ultrasonic sensor was chosen over Infrared sensors for many reasons. It is more accurate, uses less power, and is less susceptible to noise. For the physical transportation, the Turnigy 1/16 Mini rally car was chosen for its price, size, pre-installed components, and four-wheel-drive. This option was mainly just the simplest pick.

In order to begin navigation, the vehicle would have to be able to travel in a straight line. In order to ensure this, a line detection algorithm would be implemented. This algorithm was chosen for the simplicity. The idea was that the camera wouldn't use the entire image for the algorithm, it would only use a portion. It would search the image until white was found, upon this finding, there would be a set left edge of the new picture. The car should keep that along the left within a certain width. The line and the centroid of the car could be used to determine the offset of the car from the line. Along with staying straight the car would have to be able to obey signs on the road. To "read" the signs, it was decided that an ultrasonic sensor, as well as the camera, would be used. The information returned from the two could then be fed through algorithms to determine the shape of the sign. For this, OpenCV had a lot of useful functions that would be implemented.

3.1.2 Knight Sweeper 4200

The Knight Sweeper is designed to travel between two locations, and search for metal objects along the way. It could be used to find mines, traps, or improvised explosive devices so that a safe path can be made. It is all made on a four wheeled rover platform.

The Knight Sweeper directly relates to the project because it also tries to find a path between a start and end point. The path it finds is based on metal objects found along the way however, when BroBot finds a path based on any object in the way. Their project uses infrared sensors, ultrasonic devices, and a GPS unit to navigate. All three of which we could consider. Based on how well these methods worked with their project we may decide to do a similar implementation.

For this project, the team researched many popular techniques in solving their problems, but ultimately decided on a few options for their design. Due to its ease of debugging, immense amount of sample code, speed, and most importantly, the greater amount of memory, the Stellaris M3 was chosen as the microcontroller. For tracking the location, the 20 Channel SR-92 seemed the

most suitable fit. Because it is highly accurate, easy to integrate and test, it minimized noise, and it needs no additional hardware; it made it much simpler to implement. For navigation, they implemented the ultrasonic LV-MaxSonar-EZ Ultra_Sonic Sensor as well as two Sharp Sensors, which are Infrared sensors. The first is chosen because it can detect objects close and far away, and the infrared are chosen because they are economical, easy to implement, and don't require much power. All of these parts were mounted on the A4WD1. This was the final selection for their chassis due to its wide base, larger wheels, and ability to traverse through various terrains.

How everything comes together for this project is fascinating. The project is trying to find a safe path between two points by avoiding obstacles as well as IEDs, so along with the mentioned parts, there is another sensor to detect IEDs. So, the robot has the IED detector on the front edge to make sure it doesn't run over anything, the ultrasonic sensor facing forward above it, and the two Infrared sensors on the sides, slightly skewed as a way to get "peripheral vision". They then run C/C++ on the microcontroller to direct the robot's motion, being sure to follow the input from all the aforementioned sensors. Basically, if any of them is detecting something, change the direction.

3.1.3 R.C Ghost Rider

This project didn't really have a lot of motivation, other than simply being in the class and wanting to have fun. The ghost rider had two main parts; the cockpit, where the user controlled the car, watch what the car could see, and feel "realistic" motions of what the car was experiencing, the other part was the actual car that traveled around based on the motions of the driver in the cockpit.

The correlation between the Ghost Rider and BroBot basically comes down to the vehicle, video transfer, and wireless communication, though it may not even be far. This particular project doesn't have to guide itself to a destination, but it has to be able to take the directions from a cockpit fairly far away. BroBot has to be able to be called over from an unknown location to initially get him to travel towards the person in need. Ghost Rider sends the video it is continuously recording to a computer in front of the person driving in the cockpit. This can be looked into, as our project needs to be able to be able to send images to either a wireless device, or simply the processor within. This particular part of the BroBot is still being decided.

The physical body to the car aspect of Ghost Rider was just a general RC Car, this was because they didn't need to change much about the physical body for their project, and they simply needed to attach a camera and a PCB board. As for the image and data transfer, the XBee 1mW and a typical surveillance camera and receiver were used. They were the best option for the project because they had nearly identical ranges, as well as similar SNR at a distance from the cockpit.

The XBee was used to deal with the communications from the camera to display. The RC car transferred 15 bytes of packed data, as did the Cockpit, during every transmission. This happened in a serial, which is something being considered for the BroBot. Although it would have been useful, there wasn't a lot noted about how the data sent for directional input physically manipulated the RC car. It basically just talks about sending all of the data via the XBee.

3.1.4 Track Detector

Boy Scouts of America has an annual derby car race that usually requires people to take various measurements, such as the top speed of the cars, final speed, as well as track position, and many more. To assist in making the competition easier to manage, this senior design group decided to make a device that could do all of these tasks for a specific Club Scout Pack to more reliably tell the winner.

At first glance, this project doesn't appear to have any relation to BroBot, but in fact in has many attributes that could prove useful in understanding their approaches. They implement wireless communication as well as an LED display, both of which BroBot will implement. For their project, the wireless communication is used between the various subsystems, while we would need to use it between a cell phone and BroBot.

For the wireless communication a Bluetooth connection was used. This was decided because of the lack of difficulty in implementing it. For the communication of the Bluetooth device to the MSP430 processor, a SPI interface. To do this, the CC2540 that's being used acts as the master and the MSP430 is the slave, so the Bluetooth SPI input connects to the display drivers through nets in the Eagle schematic tool. For their LED display, they use it to display speeds, which is not what is of use to our group. However, how they connected the display and sent the information is useful.

To actually use the SPI interface, 16 bit words were chosen to be used. One problem that our project could encounter that theirs didn't is that we may be using the Bluetooth for more than one connection, where there's was only needed for the connection between the CC2540 and the MSP430. The connection between these two devices can be expanded to create multiple slaves, which is something we may need to consider. To send and control the display, the sensor detecting the speed was hardwired to the processor. From there the output was decoded for each of the four lanes. This implementation seems pretty straight forward, so it seems we shouldn't run into any difficulties implementing a display, as ours will not be decoding information from a sensor before displayed.

3.2 Relevant Technologies

3.2.1 Wi-Fi

Wi-Fi is a technology that has been around since the 1980s, providing powerful wireless support to all sorts of devices. Wi-Fi is a technology we are exploring to use as a means of communication between the app on the user's phone and with the item watching program on BroBot's microprocessor.

Wi-Fi is a strong technology to use as a communication method because it has a very long range. Unlike other technologies such as Bluetooth, which require both users to be within the immediate vicinity of each other, Wi-Fi allows a fast connection at distances up to 200 yards. Not only is it longer range than Bluetooth, but it operates at speeds of up to 250 mbps, allowing for us to transfer pictures wirelessly.

While normal Wi-Fi connections require an external router as a host, Wi-Fi direct is a feature allowing one of the users to act as the host, eliminating the need for the middleman. In our case, Android 4.1 and above allows for direct Wi-Fi, with additional support in 4.2. Since our requirement for the app calls for somewhere above this range, it is safe to assume all users using our app can make a connection to the processor.

3.2.2 Roomba

This product is one that's been out for over ten years now, but is still being improved by its brilliant techies. It is a little disk shaped robot that every home could surely use. The overall duty of this product is to autonomously clean the floor of a room better than a person could be the use of a broom and dustpan. Initially this product just had a few settings to pick the room size. The second generation had the ability to determine the room size itself, as well as improved dirt detection and fast charging. The latest model improved the size of the cleaning system, a new filter, as well as a better battery life. This model also used an infrared sensor to sense objects and reduce the speed. It also has a "Dock" button to force it to dock itself and charge, rather than be carried. While there were a couple models between those mentioned, a large amount of improvement happened within the years of production. Prior to the object detection used, the Roomba simply had a bumper designed to absorb the impact of crashing.

This autonomous robot has many similarities to some final goals for our project. While it must travel across every exposed inch of the floor, it still needs to be able to navigate around objects. Our project looks for the best path, but still needs some sort of object detection and collision avoidance like implemented in the Roomba. Another similarity is the traveling to the charging station. Both this

product and our project need to have the ability to travel to their charging station on their own. Knowing this product already exists makes it a little simpler and trying to figure out how we can get our project to also do this task.

In Roomba, a lot of the detection of the system is done using the infrared sensors. They start by first determining the size of the room by sending out the signal and calculating how long it takes for it to return to the sensor. These sensors are also used to find “cliffs” so that it doesn’t tumble down a flight of stairs. This part works by continuously sending out infrared signals from the bottom of the robot. If there ever is a time when the signal doesn’t return almost immediately, it knows it has found a cliff, so it backs up and changes direction of its course. For the path determination, the Roomba uses its wall sensors to figure out how close it is to bumping into something. Once it reaches the “perimeter” of the room, it simply rotates and starts going in the next direction, typically following counterclockwise rotation. This is done so that it doesn’t repeat the same portion of the room repetitively. Another reason this perimeter aspect works is that a room typically has four walls, so it is trying to travel around that. The Roomba also has the ability to set up virtual walls so that it can stay within a defined area that may not have physical walls. These virtual walls act the same way within the algorithm as the sensed walls did. For the autonomous returning to the charger, infrared signals are also used. The charger emits a signal that the robot follows to the docking location.

Knowing how the Roomba does its space detection, object detection, and overall idea behind its navigation will help come up with ideas on how to make BroBot do similar tasks. As infrared sensors are in consideration, using the same methods is a possibility. However, the user will not be emitting an infrared signal for him to follow, so the methods will only be used in the process of navigating from the desk to the user. We could also set up virtual walls so that our BroBot stays along a general path, similar to how Roomba uses them to keep within a desirable area. It states that the virtual walls send out an infrared signal, this is done because they are physical objects. However, we won’t be able to use those because BroBot is not being used in a private location. Our virtual walls will be set up within the layout of the library, if this is the decided method of travelling.

3.3 ARM Microcontrollers:

3.3.1 Texas Instruments Tiva C Series and Other Considerations:

Texas Instruments has a great selection of arm processors ranging from the very powerful to low power ARM chips. A big consideration for the arm processor is how easy is it to test with the limited resources we have as students. Texas Instruments has a great line of low priced development boards meant to act as a gateway to the ARM processor called the launch pad. At the moment only two different launch pads are made with an ARM processor, the Tiva C Series LaunchPad and the Hercules Launchpad. The general information of the

microcontroller on the Tiva C series Launchpad, the TM4C123GH6PM is shown below in table 3.3.1-1.

Flash Memory	256KB
SRAM	32KB
GPIOs	43
Operation Speed	80Mhz
Package	64LQFP
Price	5.45

Table 3.3.1-1

The microcontroller has enough speed to deal with small image processing jobs. But the RAM size is very small and would not be able to hold the amount of pictures that we want to store, but would be a good platform to start the coding and testing on with much smaller pictures so it could store them. This microcontroller can also perform floating point operations, which is what we desire from this microcontroller.

While most of the GPIOs will not be used it is helpful to have a lot of pins for testing switched and for status LEDs. Status LEDs will be important to see if a picture was sent, asked for and/or received. Also on this microcontroller is 4 I2C ports, and 8 UART ports, which is important for receiving and sending data either to the camera, the user or another microcontroller. The microcontroller comes in a 64LQFP pin package which would not be very difficult to place and solder onto a PCB even with our limited knowledge and skill. The microcontroller doesn't have the functionality to easily interface with a bank of external memory, which would severely limit the amount of data this microcontroller can access. This could cause a big problem if we run out of memory space and need to send a picture somewhere else and act like that data is in the memory of the microcontroller. Also this would be much more difficult to implement than a microcontroller that has the ability to easily interface with an external data source.

A nice feature of ARM processors is what is called micro direct memory access. This controller inside the processor can move data around while the processor deals with other operations, this could be useful since we will be using the majority of the memory coming into the microcontroller. This is extremely relevant for this microcontroller since its data space is very limited and would need to be properly sorted and moved. It can transfer data to and from the SRAM, though it since they flash and the ROM are located on a different internal bus the micro DMA cannot operate on the ROM and flash memories.

The TM4C123GH6PM doesn't meet an important memory requirement needed for this project so other Tiva microcontrollers that can interface easily with external memory where researched. TI has a feature in some of their ARM microcontrollers called external peripheral interface (EPI). EPI is an interface

dedicated for peripherals and memory and has a very large spot in the memory map. This functionally also has many different options for external memory interfacing including NAND flash, NOR flash, SRAM, and others. Therefore the TM4C129ENCPDT was looked at since it has this nice functionality. Table 3.3.1-2 below shows the general specifications of the microcontroller.

Flash Memory	1024KB
SRAM	256KB
GPIOs	90
Operation Speed	120MHz
Package	128TQF
Price	10 USD

Figure 3.3.1-2

The TM4C129ENCPDT is a 32 bit ARM Cortex-M4F processor core microcontroller. The size of flash memory is much larger in this microcontroller than in the TM4C123GH6PM, but there are some regulations when using the flash memory since it isn't perfectly EEPROM. For example only an erase can change bits from 0 to 1, also only a write can change bits from 1 to 0, and this means that there has to be a lot of care when dealing with memory operations inside the flash memory. Most ROM systems have this type of constraint, and can be overcome easily with care while dealing with the storing of the image information.

The external peripheral interface can have an 8/16/32-bit dedicated parallel bus for external peripherals and memory, which is perfect for integrating with the camera or extra RAM space. The EPI has three modes, a synchronous dynamic random access memory mode (SDRAM), a general-purpose mode, and a tradition host-bus mode. For our purposes SDRAM might me very difficult to get working correctly, but the host-bus configuration works great to access SRAM, NOR flash memory and other devices.

Unlike the first Tiva C series microcontroller that was discussed above the TM4C129ENCPDT doesn't have a cheap development board. The development board for this microcontroller has a lot of not needed features and comes in at 200 USD, which is well above the budget for BroBot. Upon further research every Tiva microcontroller that had the EPI functionally has very expensive development boards and no cheap alternatives.

The other microcontroller that has a LaunchPad and an ARM processor from Texas instruments is the TMS570LS0432 and the RM42L432. While the Hercules microcontroller is built for safety applications in mind, it still has enough functionality to do what we would require from the processor. The RM42L432 is a more powerful processor; its useful specifications are shown in table 3.3.1-3 below.

Flash Memory	384KB+8kEEPROM
SRAM	32KB
GPIOs	8
Operation Speed	100Mhz
Package	32 PDIP
Price	5.85 USD

Table 3.3.1-3

The RM42L432 has dual CPUS running in lockstep, this is to ensure that all the calculations are correct and with small amounts of error, since this processor is made with safety in mind. While this does have the muscle to do what we need, all of the special functions built into the microcontroller are of no use to this project. Using this processor would be a waste of time and effort.

Texas Instruments also has a high end ARM processor line that would be more then able to perform what is needed for the BroBot Project. These processors are ARM Cortex-A8 and A15, with over 1.35GHz clock speed, which is 10 times above what we initially wanted. They also come with easy integration of embedded Linux, which is great if we were to use OpenCV for our image processing algorithms. Texas Instruments also produces an inexpensive development board called the BeagleBone, which was created as a small and inexpensive replacement for a computer. The BeagleBone has 512MB DDR3 RAM, 2GB on board storage, 65 digital I/O 4 serial lines 2 I2C lines, and also a USB Host. This development board also has a great deal of literature dedicated to it, its use, and programming the board. A concern with using this processor is creating a PCB for the board. When discussed with other students who have gone the route of BeagleBone, they explained the cost of trying to create a PCB that could support the processor that the BeagleBone uses was well above our proposed budget.

3.3.2 STM32F407VGT6:

STMicroelectronics makes many different ARM processors for various embedded applications. They, like Texas Instruments, have inexpensive development boards with an ARM processor on it, the main difference is that the processors on St's development boards are a lot more powerful than that of what Texas Instruments offers. A good example of this is the STM32F407xx family microcontrollers, on the STM32F4Discovery kit, which are the mid-high end of their ARM processor line and upon first glance meets the general requirements for the BroBot project. Some of the more interesting parameters are shown in table 3.3.2-1.

Flash Memory	1 MB
SRAM	192 KB
GPIOs	72

Operation speed	168Mhz
Package	LQFP100
Price	5USD

Table 3.3.2-1

Also with this microcontroller comes the ability to easily interface with external memory, which is another requirement for the BroBot project. The flexible static memory controller (FSMC) has 5 different modes of operation for easy integration with external memory, there is PCCard/Compact Flash, SRAM, PSRAM, NOR Flash and NAND Flash. Since the internal advance high-performance bus (AHB) does transactions with 32 bit wide data, it will split into data that is 16 or 8 bits wide consecutively. The max frequency for synchronous accesses is 60 MHz, this microcontroller has 2 8 bit lines for the FSMC.

The STM32F407VGT6 has 4 different modes of operation including normal operation, they are sleep mode, stop mode, and standby mode. Standby mode uses the lease amount of power and would be the mode that is used when the robot is going towards its destination. The data sheet fully explains how to properly power this chip, which is great for this project since the users designing the robot are very limited in powering knowledge. The STM32F407VGT6 has 3 I2C bus interfaces which supports standard-mode (100Hz) and up to fast-mode (400kHz).

DSP instructions are also implemented into the microcontroller. The DSP instructions make the multiplying instructions executed in a single cycle, which drastically improves performance. With this functionality comes 10 times the speed of normally 32 bit floating point operations. The microcontroller also has a single precision floating point unit to help increase calculation speed of the processor.

This microcontroller also has up to 14 timers. While this high amount of timers will not be needed they will be extremely useful when trying to get a good stream of images coming into the processor. With these timers the microcontroller will be able to keep an eye on the time of how long it has been till it got its last photo, how long it took for the user to get back and how long it has been till the user asked BroBot to watch their items.

This microcontroller has built in a system to take in video and image information from a camera module or a CMOS sensor. This functionality can run 8-14 bit parallel communication between the controller and the camera module, this feature can run up to a rate of 54 Mbytes/s. While this functionality is made with video in mind it can also serve as a great tool to be used by this project. It would greatly simplify the use of the MT9D11 camera module, since this module uses 8 bit parallel data to output its JPEG images. The speed at which the MT9D11 can send information is between 6 Mhz to 80 Mhz, while we wouldn't be able to use

the max speed of the camera module the speed of this system falls within the range of the camera output speed.

The development board that has the STM32F407VG6 on it is from st.com and is around 15USD. The board itself has a header for every pin coming from the microcontroller and also has two push buttons, eight status and general-purpose LEDs. It runs on an usb connection, which also is used to program the microcontroller. Also there are many free ready to run application firmware examples on St's site. This would be useful if we were to use this microcontroller for the BroBot project.

3.3.3 ATSAM4S16B

The ATSAM4S16B is a 32-bit Atmel ARM cortex-M4 microcontroller. Shown in table 3.3.3-1 are the specs of the microcontroller.

Flash Memory	1 MB
SRAM	128 KB
GPIOs	47
Operation speed	120Mhz
Package	LQFP64
Price	11 USD

Table 3.3.3-1

The ATSAM4S16B also has and external bus interface that supports SRAM, PSRAM, NOR Flash, and NAND flash. The max IO pins on this processor is 47, and also has some DSP features built into the instruction set. It has two USARTs and two two-wire UARTs, which is all that would be needed for the BroBot prototype.

The processor has 5 different modes of operation including normal operation. The four other modes are backup mode, which consumes the least amount of power but has to wake-up before performing the immediate task, there is wait mode which is a lot like backup mode but instead the processor does not need to wake up and keeps the processing core powered throughout. The final mode is sleep mode, which stops the core clock but doesn't stop any peripheral clocks, the total power saved is dependent on which peripherals are turned off and white ones are turned off. Any of these modes can be canceled with a simple interrupt or instruction inside the software.

This microcontroller also has the capability to take in up to 8 bit parallel data, and send out up to 32-bit parallel data, which is what some of the camera modules output normally. This functionally is realized with the parallel input/output controller built into the microcontroller. The 8-bit parallel capture mode was made with CMOS image sensors in mind, which is the type of image sensor that could be used by BroBot.

Atmel has a nice development board that comes with an ATSAM4S16B chip already on it. On the board also is 2 pushbuttons, one LED a SD card socket, headers for all pins from the microcontroller, and 2Gb of NAND Flash. This development board would be very useful during our testing phase of the project. Since it already has 2Gb of NAND Flash on the board it would be easy to choose what type of external memory, if we needed it, we would get, because we would just choose the exact memory chip that is already on the development board. This board is more expensive than others that have been researched, but not too big of a price where it is outside our budget, at around 40 dollars. Also Atmel has a starter kit for this exact processor which comes with software and expansions on this board, if we were to use this board we would probably go this route.

3.3.4 Conclusion of ARM Processors:

Most of the research done for these processor was to ensure a small list of functionalities, easy implementation, enough computational power to do the image processing that this project requires, and enough communication lines needed to talk to every peripheral that BroBot is going to use for the item watcher. Another main consideration that was looked into was the size of memory and also the ease at which to integrate the microcontroller with external memory, have it be NOR, NAND, etc.

The first microcontrollers that were looked at where Texas Instruments various lines of arm processors. As it turns out these processor where either not strong enough or way too strong for the problem at hand. While their development boards are inexpensive and have a great community behind them, all the processors on those boards do not have the memory or the ability to easily integrate external memory. The higher end processors, including the BeagleBone, would cost too much during the final prototype stage due to complexity of the processors.

The ATSAM4S16B is a great microcontroller that can easily handle everything we would throw at it for this project. Its ability to easy interface with external memory is ideal and also has many GPIOs which are great for testing, and has plenty of ways to communicate with different peripherals which is perfect for the project. Also Atmel makes an inexpensive development board for this processor. But this microcontroller doesn't have built in functionally for taking in parallel data from a peripheral.

While the ATSAM4S16B doesn’t have the functionality the STM32F407VG6 does. Which is why it will be used for this project. This microcontroller was built for the processing of images and video coming in from a CMOS sensor. While we don't need that type of work it can still be used to our advantage for taking in images and video from the MT9D111. Also the development board created by STMicroelectronics is cheaper than every development board that was found for

the other microcontrollers. Another added bonus for this microcontroller is that ST has given users many avenues to explore with the STM32F407VG6 in the form of different examples for the development board.

3.4 Low Power Microcontroller

To ensure the highest efficiency possible for BroBot, we are going to use two different microprocessors. One, a more expensive, higher power processor, will be used to run the image processing program. This processor needs more memory and more speed due to the need to store images and perform operations on them in a limited time frame. The other processor is a low power microcontroller. This controller will be used for the motion software. It will control BroBot's movement, and adjust it based on destination and objects detected by sensors. In this section, we examine different options for this low power microcontroller.

3.4.1 MSP430

The MSP430 is a 16-bit microcontroller created and sold by Texas Instruments. It is a low-power, low-cost microprocessor able to run simple programs. The controller chip itself costs somewhere in the neighborhood of $2 depending on what features we need on it. It is very flexible, containing several modes to disable unneeded clocks to save power when they are not needed. The MSP430 comes in many different models, but we will focus specifically on the MSP430x4xx series. This microcontroller comes with anywhere from 4-120 KB of Flash and ROM, 256B-8KB RAM, and many other peripherals such as Op Amps, several times, multiplexers, comparators, and more.

The MSP430 has several built-in ports. It has a port for input, that is read only, and simply allows the state of the pins to be read in. It has a port for output state to be read out. It has a configuration port, a select port, and a port to enable or disable the pull-down resistor, as well as several more. We are mostly concerned with the output port. When the program decides which way to turn, or to travel straight, this signal will be reflected in the state of the output port. If it's a high voltage, it will direct the motor to run, if low, it will not.

Software development tools for the MSP430 are supplied directly from Texas Instruments. The IAR C/C++ compiler and IDE are available, as well as TI's own Eclipse-based tool Code Composer Studio.

The biggest benefit to using the MSP430 over the other microcontrollers is the cost. While the microcontroller chip itself costs around $2, the development board in its entirety only costs $4.30 and often can be received as free samples. Also, it is a very low-energy microcontroller, so it will take a load off of our power system that will be powering the microprocessor and especially movement. However, it has very little memory for a microcontroller, and has a comparably

low clock speed. While it should be high enough for our purposes, a lower clock speed could lead to clunkier steering.

3.4.2 Arduino Uno

Arduino is another kind of microcontroller used in many projects. This board is open-source, and there is a lot of documentation and use of it, making compatibility less of an issue. The processor typically comes with a development board at a fairly high price of $30.

The board comes with many nice features, such as 14 digital input/output pins, 6 analog inputs, a 16 MHz ceramic resonating clock, and a USB connection to easily load code. It comes with all the pieces needed to run it out of the box, making it very simple to set up and run. It comes with 32 KB of flash memory, along with 2 KB of SRAM and 1 KB of EEPROM. Although it can be powered from a USB connection, in our application it will need an external power source to keep it operating remotely. The software can configure each of the pins to be either an input port or an output port. The Arduino software is free, and is used to program the code for the Arduino.

Although the Arduino's development kit is priced at $30, the CPU itself is only $2.82. This is the same price as the MSP430. For testing purposes, the Arduino is much more expensive in this regard, but for prototyping the cost will remain the same. The Arduino is only an 8-bit processor, compared to the 16-bits of the MSP430. It does have much more storage space and RAM, which is likely not a factor since this microcontroller is just storing code and sensor inputs. However, it has a boot loader, which allows code to easily be loaded onto the microcontroller. The MSP430 requires a programmer device just to load code. Additionally, the Arduino controls things such as the clock speed by itself, making it much simpler to implement. For these reasons, the Arduino Uno holds a clear advantage for us over the MSP430.

3.4.3 PIC

PIC is another family of microcontrollers sold by Microchip Technology. They are cheap, widely used, and reprogrammable. It has separate memory for the code stored on the microcontroller and the data stored. All RAM locations can be used as registers to allow any variables needed to be saved. The code is saved separately on ROM. PIC controllers are widely known to have the highest speed to cost ratio, making them very effective as a low-power controller. Free emulators are available to test software before putting it on the board as part of Microchip's IDE, MPLAB. C compilers are also sold for a price that interface well with MPLAB. As students, we are eligible to receive a free version of the C compilers.

The PIC is programmable while attached to the target circuit. PICs have a feature called In Circuit Serial Programming and another called Low Voltage Programming that does not require the PIC to be removed when being programmed. This is extremely beneficial for prototyping as we will be able to reprogram the controller as needed without having to take apart the circuit, saving us from having to do extra work.

3.4.4 Conclusion for Low Power Microcontroller

After an analysis of the pros and cons of each microcontroller, we have decided on the MSP430. While the cost is attractive, the deciding factor in this decision is the flexibility. The Arduino Uno has lots of resources behind it and is easy to interface with most projects, but it does a lot of the work for the user, leading to settings such as the clock speed being uncontrollable. While it is more difficult to do all of these things by hand, it gives us a level of control over the project that we otherwise would not have. For example, if the clock speed were set incorrectly, the movement software might not update as quickly as we need and will not properly be able to detect objects in the way. The MSP430 is the best choice to support this level of flexibility.

It has been decided that an MSP430 would be the best option for our project, but there are a variety of models that can be chosen from. We needed one that would be able to be used with the specific Launchpad, which meant that it needed to have the appropriate number of pins, as well as the correct package. Along with this, it needed to be able to use the same communication styles as the ARM processor.

The MSP430 G2553 is able to be tested and evaluated on the LaunchPad Value Line Development kit, because it is already in possession of one of the team members and would save money to not have to buy a different Launchpad. This model has 20 pin DIP, as was needed. It is also capable of I2C or SPI communication. While there were a variety of models that had these two things, this one was chosen because it had the most memory available, used the correct frequency, and it is easy to obtain a free sample of it. The last reason was why it was chosen over one other model that had all the other specifications, but it wasn’t available as a free sample.

3.5 Movement

Axial steering is a common type of movement that is used in many vehicles. When it is used, the front wheels are connected by an axel, as are the back wheels. When the vehicle attempts to make a turn, it takes the front axle, if it is front wheel drive, and rotates the entire thing about the center axis of the car. This allows for a rotation of two wheels at once when the car is trying to turn. When this is done, the vehicle can take sharper turns, and the wheels move

more in unison. When trying to find chassis models that use this, it usually isn't the steering that comes with it. However, it is possible to change a vehicle to this type of steering, and it is usually considered an upgrade.

Another type of steering is skid steering. In this, when the vehicle needs to turn it breaks on the wheels on the side of the vehicle that it would like to turn. So, if the car would like to make a left hand turn, it would break on the left side of the car to allow the right half of the car to continue moving. When it does this, it causes the wheels being help still to be dragged around the turn, which would be where the skid steering title comes from. This method of steering is more common in robot chasses because it is more easily implemented, and is easier to assemble.

If it were desired to implement axial steering, since it is not typically the given implementation of steering, there would have to be a few different parts that would need to be ordered. One of these being an axial rod, which for a chassis would have to be the exact size, and the prices start around twenty dollars for the upgrade. Some other parts that are needed for an axial upgrade are the securing pieces for the axle, and the new steering mechanism. Since it is no long steered by breaking, or turning off the motors, on the wheels that need to be pivoted about, it would also need a new method of attaching the motors, as well as a different connection to the processor to control the steering.

Since this type of movement is more common among chasses and other robotic car options, it would be more likely to find an option that supports this type of steering. Since it would be more work to try to change the chassis to support axial steering, as well as more costly, and most chasses already come with skid steering, it is would be more beneficial to implement skid steering for the project. When looking for the appropriate chassis, we won't need to look for upgrade kits for the model, or a model that already implements the axial steering.

3.6 Chassis

Looking into different types of chasses, there were several things we had to keep in mind. As we received no funding, cost was pretty important. The next consideration was the physical structure and shape; this matters so much because we need to easily be able to build upon our robot. Relatedly, the size of the chassis is fairly important, as we may have quite a few parts to add onto it. The last thing that will be considered, in regards to the physical chassis selection, is where it will be used. Since it should be used in a library setting, the tires should be appropriate, and it should have the ability to easily turn ninety degrees.

3.6.1 4WD Robot Chassis

3.6.1-1 Image reproduced with permission from Hobby King

Length	180mm
Height	80mm
Width	155mm
Price	$12.19

Table 3.6.1-2

This particular robot chassis kit is sold on the hobby king website. It was designed to be used by students and hobbyists. The two red plates are built from acrylic, and contain numerous mounting points. The size allows for enough space to add various items. The motor that comes with this chassis runs at a fairly low voltage, so the question comes, would it be able to travel quickly with weight? Based on the information provided, this particular chassis, while cheap, wouldn't move at a speed we would like to achieve. Another downfall of this model is flimsy structure. There have been notes of it not being durable. While it would only be used indoors, it will be picked up and set down by the user, so this could lead to difficulties of the user dropping it. In this circumstance, we need a durable machine that won't break so easily.

3.6.2 Aluminum 4WD Robot Chassis

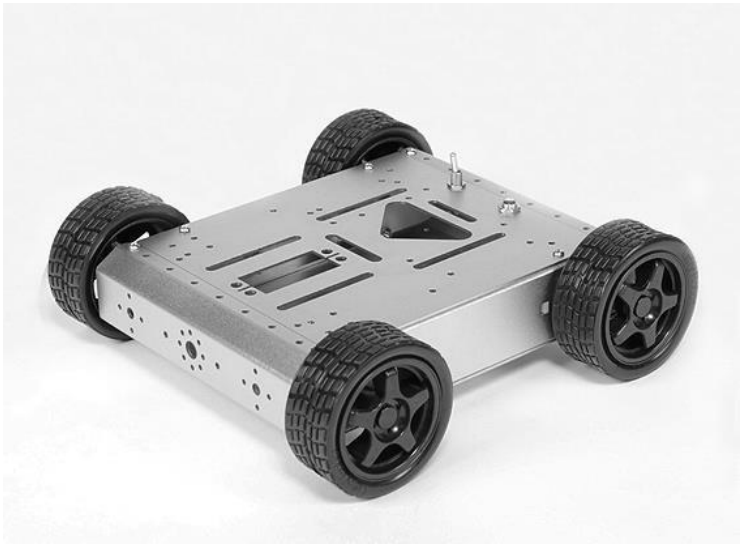


Figure 3.6.2-1 Image reproduced with permission from Hobby King

Length	210mm
Height	66mm
Width	202mm
Price	$45.37

Table 3.6.2-2

This particular robot chassis kit is sold on the hobby king website. It was designed to be used by students and hobbyists. The frame is built from heavy duty anodized aluminum with plenty of mounting locations. The internal volume was designed so that it could store a lot. The motor that comes with the kit isn't the best, but that's something that could be substituted if needed. If chosen, we may consider changing the wheels as well, as they appear to be difficult to work with. Since this frame is meant to be strong, although the maximum load is not given, it would be assumed that it can hold the weight of our attached parts.

3.6.3 Baron-4WD Mobile Platform

Figure 3.6.2-1 Image reproduced with permission from dfrobot.

Length	230mm
Height	110mm
Width	185mm
Max Load	800g
Price	$56.00

Table 3.6.2-2

This particular robot chassis is sold on dfrobot website. This chassis comes with a few interesting features. It comes with an infrared sensor switch, not an actual sensor, as well as the ease to install a look for line sensor. So, this model would be beneficial to get if we had the desire to add these sensors. It also comes with an encoder kit that is compatible with Arduino. Even if Arduino wasn't used, the encoder kit could be switched out with a different one that is compatible with whatever processor best fits the needs for our project. Obstacle avoidance would be easily implemented. The design of the body allows it to have good climbing ability. The frame on this is created from a high strength aluminum alloy. Along with this, it has thick acrylic to help reinforce areas where thin, fragile acrylic could cause defects. The motor on this model is also powerful; it can go up to 6V, which is twice as much as previous chasses' voltage capabilities. With max load, it can obtain a speed of 68cm. This model also has a platform with plenty of mounting locations of different shapes and sizes. It is also an appropriate size that would allow for the various components to be added easily. It is slightly more expensive than the last model, but it brings a lot more to the table.

3.6.4 Pirate-4WD Mobile Platform

Figure 3.6.4-1 Image reproduced with permission from dfrobot.

Length	200mm
Height	105mm
Width	170mm
Price	$49.90

Table 3.6.4-2

This chassis is also distributed through the dfrobot website. This model is intended to be used with an Arduino mobile platform, but it doesn't come with the encoders, so it would be possible to use this chassis with a different microcontroller. This kit simply comes with the frame, the motor, and the wheels. It also has a high-strength aluminum alloy body. This body, because lightweight and durable, allows for quick movement. The wheels which are included all for fast in flexible movement, even in grass, gravel, sand, or on slopes. This means that in a library setting, it would be able to travel with greater speed and precision. The electric supply voltage ranges from three to twelve volts. As visible in Figure 3.6.4-1, there are multiple mounting locations, and the size is appropriate for how much we need to add. This chassis is almost identical to the previous one, but it comes with fewer features and is slightly smaller. The features it doesn't have include the encoder kit, a power switch, the installation line, or the infrared switch mounting bracket. None of those are essential to this project though, unless Arduino is used. As an additional bonus, due to the lesser amount of features, it is slightly cheaper.

3.6.5 Dagu Rover 5 Chassis 2WD

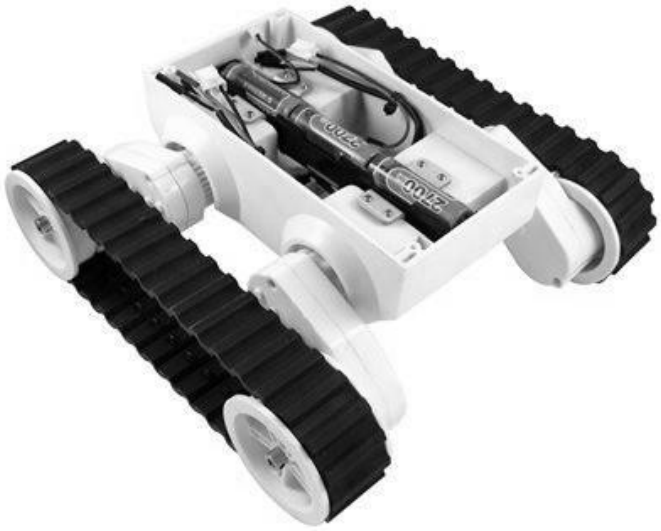


Figure 3.6.5-1 Image reproduced with permission from Active-Robots.

Length	9.5 in
Height	8 in
Width	4.5 in
Price	$80.00
Speed	25cm/s

Table 3.6.5-2

Dagu Rover 5 is distributed on a website called active-robots. The have a wide variety of different types of robots, and components to create your own robot. The model under consideration is far different than the other types of chasses; to start it doesn't have four tires. Having two tracked tires allows for easy traversal over various terrains. Within a library, it would move with ease; this would be pretty useful. The tires are capable of shifting heights to manage going over uneven ground, as well as traveling over objects. The wheels are able to lift up to almost two inches, which doubles the height of the enter frame. The internal carrying capacity is limited due to the structure of the chassis. This frame would hold the motor, which can obtain a maximum speed of 25 cm/s, as well as a few other small pieces. Another bad thing about the frame is that it would be difficult to mount an extendable camera on top. The price of this device is $80, which is far more expensive than any of the other models. The best aspect of this chassis is its adaptability of height when traveling, but even that isn't a necessity in our project.

3.6.6 Conclusion of Chassis

Based on the research of difference models of chasses, a general model was constructed of what was needed. We need a chassis that is approximately a foot in length and width, to adequately support the necessary materials, and be able to carry up to two pounds, as a high maximum. It would also need to be able to travel quickly and efficiently on carpet, while still having the ability to make sharp turns. Option 4 seemed to have the best ability to fill our needs for the project. It is a desirable cost, it is easy to maneuver, it is fast, and can easily be built onto. Something else exceptional about this model is that it doesn't require, or expect, Arduino to be used, and it will be easy to implement a different processor. It also has a sturdy body that will allow an appropriate amount of weight to be supported.

3.7 Navigation

3.7.1 Algorithm

In designing our BroBot, we had to keep in mind the method in which he should travel to the requesting user. When requesting BroBot's assistance, the user will use the app and input their location. Along this path, because there will be random obstacles along the way, he will have to be equipped to avoid collisions. Some of the methods which have been considered for this obstacle avoidance include computer vision and infrared sensors. Along with these methods, there will need to be a general path finding algorithm that will implement the aforementioned method of choice.

To start, the location of the user needs to be determined. Using a GPS to transmit the precise location to the user is something that could be useful in this. When the user is requesting the assistance of BroBot they could have the ability to allow the application to determine their location and pass the information along for navigation. This would be quite useful to be able to just give BroBot exact coordinates, but the ideal use of him would be within a library setting, and this could cause some difficulties in determining where in the building the user is, it's not as applicable as originally thought. Another negative aspect about this is the cost. Typical sensors start out around one hundred dollars, which is a little out of the budget for the project. If that weren't enough, interfacing GPS tends to be a difficult task.

Rather, to determine the location, the application could have a preprogrammed map. The map would have different labeled sections of the library, as predetermined by the group. As the user is summoning BroBot, they look at a map and highlight the area which they are located. When selected, BroBot will then have a specific coordinate within this location, probably at the center, that

he will then use as the user's location. As the traveling destination has been determined, he will then proceed with implementing his movement algorithm. This method would also be beneficiary because he will already be receiving signals from the application via Bluetooth, and this wouldn't add much work that wouldn't already be done for another part of the project. Rather, we'd be able to really implement it to its full potential.

There have been a few classes which have taught on the basics of path finding. However, all the methods that have been learned are static. The traveler tries possible paths until he finds one that works. This is done using backtracking and recursion. The traveler "looks" to all the possible directions immediately around him, if it is open he recursively calls his search on that position. Each call checks all four cardinal directions, and if the position is open, the recursive call is made to check the surrounding positions of the now current position. If at any point a position is found where it cannot travel in any of the next directions backtracking comes in , and the program considers that path null and shifts back to the last possible position and search from there. In the first instance that the traveler can reach the final destination, the path returns true and the traveler follows this path. While this method works for a layout that is given in advance, it wouldn't be able to work precisely as is within a library where the surroundings can constantly be changing. Since our surroundings are dynamic, we need to be able to regularly get feedback from the environment and move if something appears in front of BroBot. Our algorithm for transportation needs to be able to allow the robot to continuously move in a direction until an obstacle is encountered. Once there is something in the way, the robot should either change directions or just move around the object to continue along the same path.

One method that could be used for the general direction he travels is traveling with right angles along the cardinal directions. Between any two points, only two directions really need to be traveled; either east or west, and either north or south. This is true, unless the two points lie on the same path. Since BroBot will know both his starting and ending position, as well as a general layout of the library, he will know the two main directions which are required to reach the destination. So, to begin his travelling he will start traveling in one of the two directions until the two points lie along the same path. At this point, he should change directions and go straight for the user. One fault that comes from this algorithm is when BroBot needs to travel down a hallway that isn't lying in one of the cardinal directions. Another downfall is that this would not be the shortest path, and therefore would cause him to take more time traveling to the user.

Another possible method of traveling between the two points would just be travelling along the straight path between the two points, since this would be the shortest path possible. When an obstacle is encountered, it could just move around it. The same idea would be used as before, observe inputs as the path is traveled, and change direction and recalculate the shortest path to the user. So this would also account for the issue with walls or other long still objects. When

these are encountered, the robot would keep checking to see if he could turn towards where he needs to go, if he can't, he'll keep traveling along the wall until it is possible to change directions. When that point is reached, the recalculation would occur. Now the only thing missing from the algorithm is how the data will be read from the surrounding environment.

As computer vision will be using elsewhere in the project, it seemed logical that it should be considered for the traveling as well. If this method were used, the robot would need to use the camera to collect images of the environment, and decide what is actually in the path. This seems like a simple idea, because as a human we do this automatically with our depth perception, but a robot would have a bit more difficulty with this. The images would have to be used to determine all of the aspects in front of him; the walls, ceiling, floor, tables, chairs, people, etc. BroBot would have to know that it's okay to travel on the floor, but if there is anything directly in front of him, he would need to move. This now plays into the depth perception; one idea would be to just use the size of the object to determine if movement is necessary, but this wouldn't necessarily work if there were a book shelf in front of him. It would be a large object, that may need to be traveled around, but BroBot could take it as being closer than it really is. Another issue is that a camera wouldn't be able to see anything out of range, such as a person walking towards him or an object that is beneath the camera's line of sight. While these things could then be fixed with collision recovery, a person might get annoyed if BroBot ran into their foot because it was out of his line of sight. After these things were found, computer vision seemed far too complicated to use as the method of observing the environment.

The use of infrared sensors is another possible implementation of the object detection during the traveling. IR sensors are typically passive, which means that it only reads infrared signals, rather than emitting them as well. This could be implemented within the motion algorithm by determining when there are objects within too close of a range. Rather than sounding an alarm, which is typical for most uses of PIR sensors, it could redirect the signal of the detection of an object to the movement algorithm and tell it that the direction needs to change.

For the final decision on the algorithm that should be used, not just one of the mentioned methods would suffice. Instead, Bluetooth will be used for the determination of the user's location because it is already being implanted by the system, and it isn't too difficult to implement. The robot will use the location of the user and have a general predetermined route around the large objects and walls along the way. While traveling the route, it's likely that there will be objects in the way, such as moved tables or chairs; for this the infrared sensors will be used to divert the path away from the object, just long enough so that it knows it is no longer in the way. From this moved location, BroBot will adjust his path based on his location on his preprogrammed map and continue towards the final location. To know the location on the map, there will be internal calculations done using the speed, amount of time traveling, as well as the direction being traveled. So

the actual algorithm that is used will use parts of the path finding algorithm previously mentioned, as well as interrupts from the infrared sensors.

3.8 Sensors for Navigation

3.8.1 Sharp GP2Y0A02YK0F

The Sharp GP2Y0A02YK0F is a long range IR sensor. The package contains an infrared emitting diode (IRED) and a position sensitive detector (PSD). It can detect objects between 20 and 150 cm away. The closer an object is to the detector, the greater the output of the package's signal processing circuit. This signal can vary between 0.5 V for a distance of 150 cm and 2.7 V for a distance of 15 cm.

Feature	Specification
Measurement Range	20 to 150 cm
Analog or Digital Output?	Analog
Size	29.5 x 13 x 21.6 mm
Typical Current Use	33 mA
Supply Voltage	4.5 to 5.5 V
Price	$14.95 each

Table 3.8.1-1

A concern with the IR sensor is the color of the reflective surfaces. According to the data sheet, if reflected from a white sheet of paper, 90% of the originally emitted IR signal's intensity will reach the detector. If reflected from a gray sheet of paper, only 18% of the original signal will reach the detector.

The long range sensor is not suitable for BroBot. BroBot is a compact robot and does not need to adjust its path for obstacles more than 60 cm away. 60 cm is enough space for BroBot to stop and change directions. A short range device would better suit our needs

3.8.2 Sharp GP2D120XJ00F

The Sharp GP2D120XJ00F is a short range IR sensor. The detector can find objects between 3 and 30 cm away. Similar to the Sharp GP2Y0A02YK0F, the proximity to the detector increases the amplitude of the output signal. This would be useful in a project that requires a variety of actions based on distance.

Feature	**Specification**
Measurement Range	4 to 30 cm
Analog or Digital Output?	Analog

Size	29.5 x 8.4 x 13.5 mm
Typical Current Use	33 mA
Supply Voltage	4.5 to 5.5 V
Price	$13.95 each

Table 3.8.2-1

The short range sensor is more applicable for BroBot. Unfortunately, the maximum range of 30 cm might not provide enough space for impact-free maneuverability. A device that can have a greater reach but still detect obstacles less than 10 cm away would be preferable.

3.8.3 Pololu 38 kHz IR Proximity Sensor

This device is an IR sensor with a short range. The package contains both an IR LED and an IR detector, the Vishay TSSP77038. The LED can be turned off and on using an I/O pin from the microcontroller. When the device is powered, the LED is set high by default; as long as the Pololu is ON, the LED will emit an IR signal. Unlike the other two sensors, the Pololu has a digital output. This will allow us to directly communicate with the microcontroller without any conversion of an analog signal into digital. The device will simply output low when it detects any IR signal.

Feature	**Specification**
Measurement Range	30 or 60 cm
Analog or Digital Output?	Digital
Size	10.16 x 5.08 x 15.24 mm
Typical Current Use	16 mA
Supply Voltage	3.3 to 5.5 V
Price	$4.95 each

Table 3.8.3-1

The Pololu is the best choice for BroBot. The digital output and enable make integration simple. Also the voltage needed to be supplied to this sensor is in the range of most of the other items that will be used by BroBot. This consideration will also only take up two pins on our microcontroller. Since we are using a low power low pin out microcontroller

3.9 Camera

3.9.1 Camera Setup

The final aspect of constructing our robot is deciding how the camera will be set up on the chassis to allow for maximum item watching capabilities. BroBot should

be able to see all the items on the table, but whatever means necessary. The camera is small and lightweight, so the structure doesn't have to be extensively strong or large to support it.

Considering first the raising of the camera, we must lift it a distance that will allow BroBot to peer over the edge of the table, as it seems most fitting that he should sit in the chair to take the user's place when they need a break. Some different materials to consider for lifting it are PVC pipes, a thin metal rod, and a rectangular piece of wood. Starting at the beginning, PVC pipes are lightweight, sturdy, the size can be adjusted as needed, and they're cheap. Along with this, it is hollowed out so the cords would be able to be placed inside to hide them from view, as well as protect them. Next a metal rod would be sturdier than PVC pipe, but it would also weigh more, which could cause problems with the chassis, as it will just be creating unnecessary weight, but it would also be hollowed out to allow for cord protection. The last option would be a solid block of wood. This option would be the sturdiest, as well as being easier to attach to the chassis, but it wouldn't allow for the cords to be hidden. This being said, PVC pipe would probably be the best option for material to use.

The next thing to consider is how the camera's position will be able to be shifted after BroBot has been placed in the chair. One option would be simply rotating the pipe the camera is connected to. This would result in a wide range of horizontal movement, but the camera wouldn't be able to be tilted up or down, if it were needed. Another flaw in this approach is that it would be more difficult to connect the rod to the chassis and still allow rotational movement. Another option is attaching the camera onto an extension that is added on the end of the pipe. Luckily, PVC pipes have a lot of options for extension. To obtain the various pitches, a hinge support could be used, but this alone wouldn't allow for changing the yaw of the camera. While a hinge would be nice, it could be hard to find a hinge support for PVC pipe, so instead, we could use an angled connecting piece to change the direction; if two were used we could allow them to rotate and simulate the hinge movement. Lastly, we need to take care of the horizontal movement. We could add more extending pieces to the PVC pipe, but this would cause the weight to begin to be more unevenly distributed. Another option would be a ball and socket type joint, this would allow for horizontal movement, as well as some vertical, so only one angle connecting piece would be needed; or a hinge support if one can be found for PVC pipes.

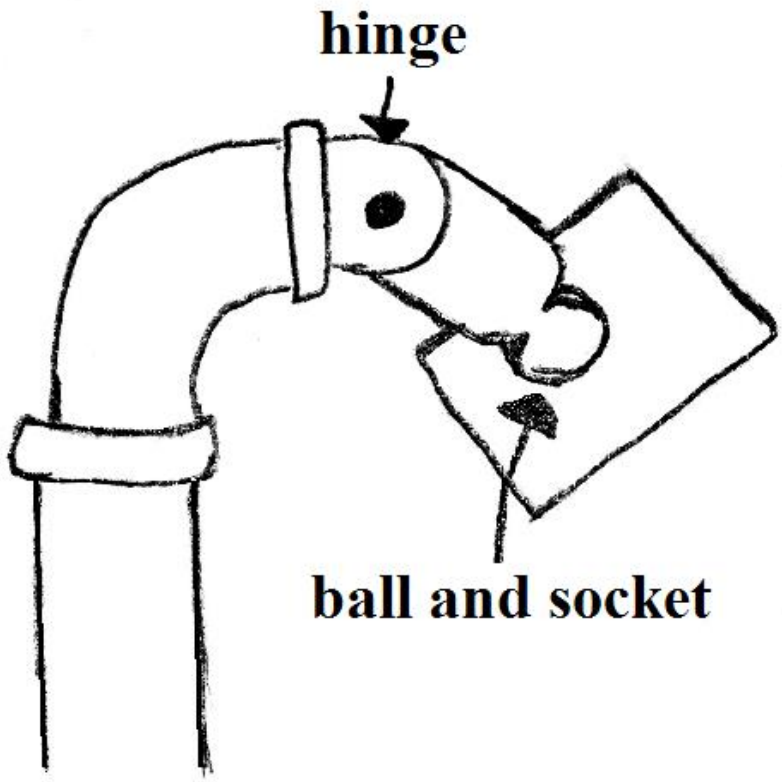


Figure 3.9.1-1

3.9.2 JPEG Image/Video Compression:

Since raw image files would be very large and hard to work with, the JPEG standard of image compression was looked into for this project. JPEG is a very robust type of lossy image format, which has a selection of quality which affects the overall size of the picture. For a high quality JPEG can reach 10:1 size adjustment from a raw file to a jpeg image. The lower you go in quality factor of the JPEG image the more image artifacts you gain due to the compression. Compression artifacts could have a great effect on our project if they start to interfere with the image processing. By a quality factor of 10 artifacts start to overtake the boundary of the object inside the image. For our project we will need to use a quality factor higher than 10. But with a quality factor of 25 the objects in the image almost have no real lose to them. The range of 10-25 will be used for the project when deciding on the quality factor. Table 1 below shows the size of the file with different values of quality, these numbers are from Visengi's JPEG encoder, with a picture at 352*288 resolution, which is much smaller then what might be implemented into BroBot. It also must be noted that this is only to get a general idea of how well JPEG compressed depending on

what the quality factor is, these numbers will not be true at all for our implementation.

Quality Factor	Compression Ratio	Quality Factor	Size(Bytes)
100	1.62	50	12.89
95	3.53	40	14.86
90	5.03	30	17.76
80	7.45	20	22.99
70	9.48	10	35.49
60	11.27	1	68.36

Table 3.9.2-1

Table 3.9.2-1 shows how well the JPEG standard scales with the quality factor. Looking at this data shows that there is a pretty big increase in compression ratio near the end of the quality factor. While a quality factor of 1 would be the smallest it would not be useful when trying to identify different objects the user wants watched. But around the range that was talked about above the compression ratio is still very good when compared to higher qualities. This data also shows that from a quality factor of 50 to 90 there is little change in the compression ratios.

Also in the jpeg standard is the ability to change the chroma components. These components deal with the amount of data that is used to describe the color of the image, and since not a lot of color is needed when watching the items it's another tool to help scale down the image size. Below is data from Visengi's JPEG encoder when it comes to the compression factor of using different chroma schemes, this does not show the monochrome option in the jpeg standard, and might be available in the camera module that will be used.

Chroma Subsampling	Bits per Pixel
4:4:4	3.8
4:2:2 Horizontal	3.2
4:2:2 Vertical	3.2
4:2:0	2.8

Table 3.9.2-2

From table 3.9.2-2 we can see that the bits per pixel decrease slightly when using a smaller chroma subsampling. For this project the least amount of sampling should be implemented for the system to ensure that the picture is as small data wise as possible. These numbers are also don't tell the full story of how the subsampling shrinks the size of the final image. Since JPEG also encodes the color sometimes the picture could be smaller if a higher subsampling is used, but this normally isn't the case.

Another way to scale down an image is by lowering it resolution. This can dramatically lower the amount of information that is need to be stored for an

image. For example going from 800x600 resolution to a 1900 x 1200 increases the amount of pixels by a factor of 4.75. Though this problem cannot be explored during primary research and must be looked further into during the testing part of the prototype cycle.

A problem that might occur with using the JPEG format is that jpeg is encoded using entropy coding. Entropy coding uses a zigzag route to arrange the image components inside the file. This might make the programing of the item watcher more difficult to write then it would for a raw file that uses a horizontal/vertical encoding scheme. But this type of encoding is done throughout the standard, so as long as the pictures are the same resolution there shouldn't be too much of a problem, as long as there is care in the program.

JPEG is one of the most used formats in embedded applications and in the world. Because of this many modules exist with cameras that output the jpeg format. No other form of compression was looked at for this project since JPEG is so widely used. Also there is a standard for video that uses JPEG encoding. Video can still be explored for this project if there is enough time to fully test it.

3.9.3 IR Motion Sensor

In general, infrared sensors are those which can see the light emitted within a certain range or the spectrum, which is naked to the human eye. They are capable of detecting heat as well as motion. Different sensors can detect different length signals; the ranges go from 210 nanometers up to 100 micrometers. Within these, they are classified as either near, mid, or far. There are two types of IR detectors; active and passive. Luckily our project is designed for indoor use because IR sensors become faulty when there is too much humidity.

Active sensors emit a signal from either a light emitting diode (LED) or a laser diode. This signal is emitted then reflected back to a receiving diode and produces a signal. These types of signals are able to provide count, presence, speed, and occupancy. Due to the advanced amount of things determined by these, they would surely be more difficult to implement, not to mention that none of those things are needed to be determined.

This is the type of IR sensor which is used in the Roomba. The other type of IR sensor is the passive infrared sensor. These simply pick up on the infrared signals put out by objects within the detectable region. These have been used in many applications, such as house alarm systems, computer mice, television remotes, and most importantly, autonomous cars. Passive IR sensors are a little simpler than active IR sensors, and still accomplish the tasks this project needs them to, so they will be used.

Objects at room temperature emit radiation between the range eight micrometers to twenty-five micrometers, which is classified as mid-range infrared. The human body is slightly warmer than room temperature, so it begins border lining the far-ranged infrared. Since the majority of the objects that will need to be detected are inanimate, and will be at room temperature. The mid-ranged infrared signal should be used. Since it will be on a vehicle, multiple mid-range PIRs will be used.

3.9.4 TTL Serial JPEG Camera:

The TTL Serial JPEG color camera from adafruit.com is a small CMOS image sensor camera that outputs a JPEG serially through 3 wire UART communication. The module itself is very small at 32mm x 32 mm. The camera also can be used as a video camera with a frame rate of 30 fps in 640*300 resolution. The largest jpeg resolution that this camera can output is 640x480 which is small when compared to the other camera modules, but is alright for our desired ability of the image watcher process.

There are some other nice added features to this camera system. One is the ability to manually adjust the focus of the camera. This feature can be changed so that the camera is first in auto focus, but if the user wants to use the manual focus it would be integrated into the application, focusing would be beneficial to the user if they wish to only watch items that are in the background of the image. It would bring an added dimension to our project that was overlooked in the initial project description.

Also built into the camera system is a motion detection option. This option would bypass the need for an arm processor and would drastically simply our design process of BroBot. Though this option is very nice for out project the user doesn't have total control of the motion detection, which could cause some misfires of the callback system. Also not having complete control means that there is absolutely no way to tweak the system when it comes to motion detection, which could cause many problems down the road.

The range of the camera is about 10 to 15 meters which is adjustable with the lens. The module runs at 75mA and with an operating voltage of 5V DC. The transistor to transistor logic runs at 3.3V. If we were to integrate this camera system with the proposed BroBot system it would take at least 5 wires, with 3 wires for the serial communication and the other two wires for powering the camera system. This number is much less than the MT9D11. The baud rate at maximum is 115200 which is considerably slower than the MT9D11, this is mostly due to the fact that the image that will be transferred will be transferred via the three serial wires instead of 8 parallel data lines that the MT9D11 uses.

This module was created for easy integration with an Arduino microcontroller system. It is intended to send a picture to the microcontroller and then the microcontroller sends the data wherever it is needed, for our project this will either be the SRAM that will store our photos or the data space on the ARM processor. While this complicates the system by one microcontroller, it also helps that there is a preexisting projects that have used this camera with the Arduino.

3.9.5 MT9D111:

The MT9D111 is a device with a 2 mega pixel camera, a processor to control the data from the camera, and different encoding features as shown in the figure below. The module that has this device on it from uctronics.com uses a 2 wire serial interface to control the system's different modes of operation. The camera can output different types of image file including JPEG and can also output a video stream. There are 8 data lines coming from the camera which has to be interfaced to a microcontroller.

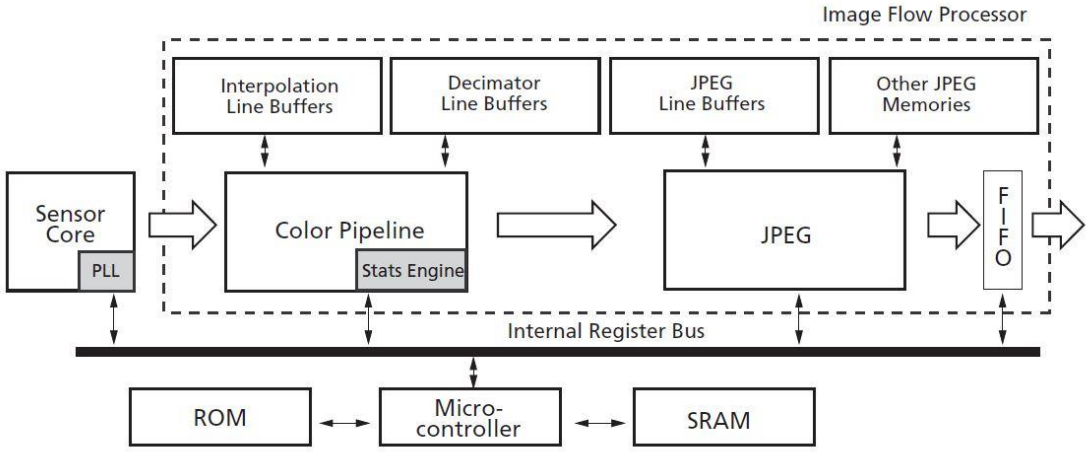


Figure 3.9.5-1 with permission pending

The MT9D111 has great low-light performance which is perfect for use inside of a library, which doesn't have a lot of outside light in it due to its size. The size of the module is very small and has 20 header pins coming out from the board. Also if we wish to have the camera mounted above the body of the robot then it wouldn't be much of a problem since the MT9D11 is very light weight and made to be held up by its PCB.

The maximum resolution of the camera is 1600 x 1200 pixels (in 4:3 format), the pixel size is 2.8 um x 2.8 um. The maximum data rate that the module can output is 80 MB/s since its master clock can range from 6 Mhz to 80 MHz. At full resolution the camera will run at 348mW though for our purposes the camera will

not consume this much power, but is a good reference when selecting our power components.

The two line serial port gives you access to the all the functionalities of the MT9D11, including things like zoom, gamma and contrast. You also can set up the specific mode that you want to use and put the system into standby mode, which is used to save power.

One of the more useful features of this system is its ability to have full control of the JPEG encoding. It has the quality factor tables so we can have full control over the quality of the image, which would be useful during testing to see how big the files are and seeing how fast we can process though images and adjust accordingly, even though quality doesn't linearly scale with the data outputs it still can have a major factor in the size of the file. It also has three different color schemes for the JPEG images that are outputted by the system, 4:2:2, 4:2:0, and monochrome. With 4:2:2 having more colors available this also helps scale down the size of the file that will be outputted. Also the system is made to have a stream of 30 fps of JPEG images, which is much faster than what we intend to implement in our system.

This camera system can also output the raw black and white image files, which could be useful if we run into trouble with the image processing on the JPEG images, i.e. amount of calculation power is too large for our processor to handle. While these files might be bigger since they are not compressed they would be easier to work with then JPEGs.

In the camera module is an auto focus feature that uses a focus algorithm that will try to maximize the sharpness of the vertical lines. This takes out a lot of guess work from our side on how to correctly use this camera when it is interfaced with the microcontroller, meaning its one less thing that we will have to worry about when it comes to getting the system to work if we do indeed choose this system.

While this system does have many nice features that could help us to compete our task and also help during the testing of BroBot the data out could cause an interesting problem. This problem is that using this module will take a minimum of 13 lines that tell the microcontroller when a line and frame is valid. Therefore if this camera system will be lines, 2 for the serial communication, 8 for the data lines, and 3 more handshaking implemented in our system we will need enough pins on the microcontroller to interface correctly with the camera. Another consideration for this system is that the data sheet and developer guide leaves a lot to be desired. Unlike the other camera that will be discussed there isn't too much information on other projects using this specific camera. All the information from other people is that this system has a lot of strange errors and might output something that was not intended by the user. Finally the cost of the module of this system is quite low at about 20 USD.

3.9.6 Conclusion for cameras:

For the image watcher the camera is gateway to the outside world. The camera needs to be able to adjust the picture coming into the system so it will be easier to fix the system if there is any problem of performance on the main arm processor. For BroBot the best answer isn't apparent when looking at these two cameras, while the MT9D11 is very powerful and also very robust the TTL serial JPEG camera from adafruit.com encloses a lot of what the project is doing. Since this project is a senior design project, where the group members want to learn as much as possible about different applications of image processing, the MT9D11 best suites our needs.

MT9D11 was picked because of its ability to send a lot of data very fast and also for its robustness. Since the camera system can control the quality factor of the jpeg that will be sent out we will have great control on the amount of memory the image will affect. The smaller the size of the image bit wise the less amount of processing time will be needed, therefore increasing the performance of BroBot. Since a limiting factor on the speed of processing is on the microcontroller which once decided on will be hard to change, it is a great thing to be able to change the size of the picture coming in.

3.10 External Memory

Since the pictures might take up more space than is available on the microcontroller some type of external memory will need to be interfaced, both extra RAM and ROM will be discussed. Since the STM32F407VG microcontroller was chosen earlier in the report the memories that interface well with that microcontroller will be looked into. Since the size of the internal SRAM on the microcontroller is can only hold a couple pictures along with other things Alliance memory's AS6C62256 was looked into. Figure 3.10-1 shows the overall information of the SRAM.

Size	256kB(32K x 8)
Package	28-DIP
Interface	Parallel
Voltage Supply	2.7V - 5.5V
Speed	55ns (18MHz)
Price	1.79USD
Memory Type	Asynchronous SRAM

Figure 3.10-1

This SRAM was chosen mostly for its ease of testing since it has a 28DIP package. A problem that can arise from using this memory is that it is much slower than what the microcontroller can interface with. This could cause a

bottleneck effect of the flow of data. This particular unit is very cheap and would interface well with the microcontroller since it takes in parallel data. Also the voltage requirements for this IC are very wide, which is extremely useful. It would lower the amount of voltage leads we need when designing the power portion of the project. If we wanted more speed we would need to use an IC that consumes more power than the AS6C62256. Below in figure 3.10-2 are the specs for IDT's IDT71256SA/TTSA SRAM.

Size	256kB (32K x 8)
Package	28-DIP
Interface	Parallel
Voltage Supply	4.5-5.5V
Speed	20ns (50MHz)
Price	3.82 USD
Memory Type	Asynchronous SRAM

Figure 3.10-2

This IC is much faster than the previous SRAM IC, but that comes at a cost. That cost is the voltage that is need is much higher and the memory will use more power. Those are the two main differences between these IC's. For SRAMs with dip packages this is the highest amount of information the SRAM's can store. If a SRAM that doesn't have an easy package is used then a breakout board will need to be bought and installed. Luckily these memory packages would be perfect for our project if more SRAM is found to be needed during testing.

While more SRAM would be nice, it would be more important to have more overall memory to store a good amount of images. This is so the microcontroller can keep a record of all the pictures that were taken and would be able to send any of those pictures to the user or security. This type of data storage is best for ROM devices, since it doesn't need to be extremely fast but it does need a lot of space. The microcontroller can easily interface with NOR and NAND flash, therefore those types of flash ROM was researched. Figure 3.10-3 shows the specifications of Spansion's S25FL512SAGMFI011.

Size	512Mb (64M x 8)
Package	16-SOIC
Interface	SPI Serial
Voltage Supply	2.7V-3.6V
Speed	133MHz
Price	6.93 USD
Memory Type	NOR Flash ROM

Figure 3.10-3

512 Mbytes would be enough space to store many pictures even at a high resolution and no real compression. A breakout board would need to be bought if

this memory IC were to be used, which would increase the price by about 5-10 dollars. Though this does have a real fast speed it only takes serial interfacing for information. While this won't cause a problem it would mean that communication between them would be very slow. It would be increasing difficult to test with a memory IC that could not be put on a breakout board. This severely limits the amount of memory units that could be used on this project. One more memory ic was explored. Figure 3.10-4 shows Winbond Electronics' W25Q128FVSIG-ND.

Size	128Mb (16M x 8)
Package	8-SOIC
Interface	SPI Serial
Voltage Supply	2.7-3.6V
Speed	104MHz
Price	2.76 USD
Memory Type	Flash ROM

Figure 3.10-4

The W25Q128FVSIG-ND is an inexpensive serial flash IC that would meet our memory needs for this project if our memory needs turn out to be small. The main difference between these two IC's other than one is NOR and the other is NAND flash ROM, is the overall price. Though the price difference is within our budget it would be beneficial to use the smaller memory size, since it is only serial communication lines that need to be interfaced with the microcontroller.

A main problem with this using this external memory unit is that it would not use the flexible static memory controller in the microcontroller. This is because the controller is made for parallel interfacing with external memory units. The last external memory that will be looked at is microchip's SST39SF040-70-4C-PHE.

Size	4Mb (512Kb x 8)
Package	PDIP-32
Interface	Parallel
Voltage Supply	4.5-5.5V
Speed	70ns (14 MHz)
Price	1.90 USD
Memory Type	NOR Flash

Figure 3.10-5

Figure 3.10-5 shows all the specifications for the SST39SF040-70-4C-PHE. While this is the smallest memory that has been looked at, it is fully supported by the microcontroller's flexible static memory controller, which is will be very useful. Also this comes in a nice dip package so it will be easy to test with. Which is what is needed for this project. The size is just above the size needed to hold a minute of passed images.

To come to a conclusion on which external memory IC will be used there needs to be some testing done on the camera and the microcontroller that will be used. The testing will need to look at the size of the picture coming from the camera, this will give a much better picture of what is needed. For example if the pictures are small but the program needs a lot of RAM to do the needed calculations then one of the SRAMs will need to be interfaced with the microcontroller. If the pictures are larger than anticipated (over 30 Kbytes) then both memories will need to be interfaced, or a completely new microcontroller might be needed. Research into implementation shows that these memories would be difficult to implement with the microcontroller without a good breakout board. Since it isn't yet known if external memory will be needed there will be no external memory interfaced, but the research is there.

3.11 Android Application

3.11.1 Communication

In order to enable communication between the information generated by the item watching program on the microprocessor and the user interface on the Android app, a channel must be maintained between the two devices. Several such technologies exist. Some, such as a USB connection, are wired and will not work for our purposes. Others, such as infrared, are rare on platforms such as Android and often require direct line of sight in order to operate. A third category including sonar requires complete lack of noise from the surrounding environment, a situation not likely to happen on a college campus (or anywhere else). We will focus our attention on two of the most widely used forms of wireless communication, Bluetooth and Wi-Fi.

Bluetooth is a technology created in 1994 to replace wired RS-232 serial communication busses. It is highly secure, and operates up to a distance of about 110 yards. It is very universal, and comes standard on almost every Android phone imaginable. Tablets are less common to have it, since it's used often to connect phones to car systems. It is also very cost effective – a common Bluetooth module for a microprocessor normally runs for around $8. However, our requirements demand as high a communication radius as possible, and Bluetooth simply cannot operate at long enough distances.

The other major technology we are examining is Wi-Fi. Specifically, a form of Wi-Fi called Wi-Fi Direct. Typical Wi-Fi operates from a device acting as a router, with all users of the connection accessing the router. Direct, on the other hand, allows one of the devices involved in the connection to act as the router and cut out the middleman. It maintains Wi-Fi speed and range of up to 200 yards at 250 mbps with even more universalism than Bluetooth. The downside to this technology is that it's expensive. A typical Wi-Fi module costs around $50 to

$100 to buy. Additionally, Wi-Fi is not as simple to implement, and will not provide the same ease of integration that Bluetooth will.

3.11.2 APIs

In the world of computer vision, two major application programming interfaces (APIs) are used. These two are OpenCV and OpenSURF. Both of these contain libraries that can be included in programs running on most major operating systems that enable the use of many useful functions for computer vision.

OpenCV is a library that contains functions useful for all sorts of vision problems, from facial recognition to augmented reality. It also contains an entire library that can be included to allow the easy use of machine learning. Basic functions such as cvLoadImage and cvShowImage are used to read images into the program and display them. More advanced functions useful for this project include the object detection functions, namely CascadeClassifier and HOGDescriptor, both of which can be used to detect common objects using machine learning. For this project, we could detect items such as notebooks, laptops, and backpacks by using machine learning concepts, and then track these items to ensure they don't disappear. Another possible approach is using OpenCV's matching keypoints functionality to find similarities between two images. Using this feature, we can compare each new picture taken against the original, and check it for differences. A variety of this method is what we will be using and is detailed in another section.

OpenSURF is a slightly lesser known API that has a different set of features to OpenCV. OpenSURF contains a very useful feature called background subtraction that allows the computer to differentiate between the background and foreground of a picture. That is, the programmer must specify what consists of the foreground, and what consists of the background. Items in the background are ignored for changes such as lighting and brightness changes. Another nice feature of OpenSURF is blob detection, where a point can be extended to its boundary to detect the entire object. This feature could be used when the user chooses a point on their object to watch. Blob detection could expand that point to the entire object and could then use item tracking functions to monitor this blob.

An API that will be used in the Android app is the Android Bluetooth API to establish communication with the item watching program. Importing android.bluetooth allows the application to implement Bluetooth features, establish a link to the microprocessor via the Bluetooth module, and exchange data with it. The getDefaultAdapter() method returns the Bluetooth radio equipped on the user's phone. If this returns null, the user does not have Bluetooth and we know we can exit the app with an error. Following this, the API contains a method to check if Bluetooth is enabled on the phone. If it is not, the user can be prompted to turn it on directly from the app without having to

navigate to the system settings. Once Bluetooth is enabled, methods exist to check if the Bluetooth module on BroBot is already known – if it is, it is simply used to connect, if not, it is discovered and added to the list of known devices. The ARM processor used for the item watching program will also need to include this API in order to manage its end of the connection process and have a way to send out data.

3.11.3 App Picture Manipulation

The app must deal with pictures by displaying them for the user to see. The item watching software has the functionality to send pictures of the user's items to the app for an added sense of security and also if something is stolen. Several special functions must be called in the app in order to display pictures.

```
ImageView  imageView =
new ImageView(getApplicationContext());                    (1)
LayoutParams lp = new
LayoutParams(LayoutParams.WRAP_CONTENT,
LayoutParams.WRAP_CONTENT);                                (2)
String path = Environment.getExternalStorageDirectory() + "/your folder
name/image_name.bmp";                                      (3)
Bitmap image = BitmapFactory.decodeFile(path);             (4)
imageView.setImageBitmap(image);                           (5)
RelativeLayout rl =
(RelativeLayout) findViewById(R.id.relativeLayout1);       (6)
rl.addView(imageView, lp);                                 (7)
```

These commands are integral to displaying a picture to the app. After the picture is received from the Bluetooth link, it will be stored in local media. (1) simply initializes an imageView given the phone's inherent screen properties. (2) creates the layout parameters and accepts the settings to use as parameters. (3) creates the path location. It gets the phone's storage space from the phone itself and appends the directory that the pictures are saved to. (4) retrieves the file from memory and stores it as a bitmap, using the path that we created as a parameter. (5) uses the imageView we created in (1) and sets the picture to the bitmap. (6) just creates a layout of the screen, upon which the imageView containing the picture is added in (7).

To get the picture onto the phone from the item watching software, the Bluetooth link must be utilized. Once the devices are connected, the BluetoothSocket can be used to invoke the connection. Once set up, the socket's inputStream and outputStream may be used to share files

3.12 Voltage Regulators

Since we need a 3.3V line to power all of the IC's on the board a couple of different linear regulators where looked at. Some considerations for these regulators was the ability to pass up to 200mA, though our system might not need to pass this much current through the system we should have some wiggle room built into the system. Since the battery isn't known yet the input voltage on the regulators need to be within a good range. From around 5-10V should suffice.

3.12.1 LT1121CN8-3.3

The LT1121CN8-3.3 is a linear regulator produced by linear technology the table below shows the needed numbers of the regulator.

Input Voltage Range	4.17V - 30V
Output Voltage	3.3V
Operating Temperature	0C - 125C
Package	8 - PDIP
Voltage-Dropout(Typical)	.42V @ 150mA
Max Current Output	150mA
Price	3.02 USD

Table 3.12.1-1

This is a great linear amplifier that uses an 8 PDIP package for simple integration into the PCB. The implantation is very simple and has a shutdown option, which could be useful if the system runs into unexpected trouble. The capacitor that would be needed is a 1uF capacitor. While having an 8 PDIP package would be nice there are other regulators that don't have 8 pins that need to be interfaced.

3.12.2 LT1587CT-3.3

The LT1587CT-3.3 is a 3 prong linear regulator produced by linear technology. The table below shows the numbers of interest for this regulator.

Input Voltage Range	>=4.75V
Output Voltage	3.3V
Operating Temperature Range	0C - 125C
Package	3 Prong, through hole
Voltage-Dropout	1.15V @ 3A
Min Current Input	3.1 A
Price	6.16 USD

Table 3.12.2-1

This regulator will need a lot of amps to be able to use in our system. Though it does have 3 prongs, which would make implementation simpler. Also this regulator doesn't have a cap on the input voltage, which could be useful depending on the battery that we use.

3.12.3 LM2594N-3.3

The Lm2594N-3.3 is a switching regulator made by Texas Instruments, it ensures a +/- 4% tolerance on output voltage under all of the given range of input voltages. The table below shows all of the needed information of this linear regulator.

Input Voltage Range	4.5V - 40V
Output Voltage	3.3V
Operating Temperature Range	-40C – 125C
Package	8-DIP
Synchronous Rectifier	No
Output Current	500mA
Price	3.02 USD

Table 3.12.3-1

While the LM259N is inexpensive and comes in a nice package there is a major problem if we plan on using it. And this is the current coming out of the IC is very high and could cause problems for our system. The temperature range is really nice but we will more than likely go with a linear regulator instead of a switching regulator.

3.12.4 Conclusion for Linear Regulators

For this project we will use one LT1121CN8-3.3 linear regulator since it meets all of our requirements nicely. This regulator is cheap and doesn't need a large amount of current to regulate the voltage. Also since it is a 8-PDIP package we can put a socket on the PCB for this IC, just in case we cause physical harm to the device by pulling too much current. Another nice feature of this IC is its price at only 3.02 USD

3.13 Bluetooth Modules

For our system, the Bluetooth module needs to satisfy a few basic requirements. To be able to communicate with our processor, the Bluetooth that is chosen should be able to use UART serial communication, as well as obtain the farthest connection possible, as decided by the different classes of modules. There are three classes for them; class 1 has a range of 100 meters, class 2 has a range of 10 meters, and class 3 has a range of 1 meter. The chosen device will also need to be able to interface with an android device, which shouldn't be too difficult

because Bluetooth devices come with their own address that an android phone should be able to just look up and connect to. It should also consume low amounts of power. Based on the size of the chassis that will be used, the Bluetooth should be as small and lightweight as possible, while first satisfying the other requirements. When reading the datasheets that accompany the devices, we should be able to understand them so that we can more easily work with the device, this means they need to be in English. Along with this idea, we need to be able to easily test the device by soldering wires to it to test its capabilities. Lastly, the price of the device will come in to effect because our budget is limited.

The first module under consideration is the TI LMX9830. This model was made to communicate with UART serial communication, and is capable of being interfaced with Android devices. Another positive thing about this model is that it consumes very little power and is fairly cheap, but in the description it is said to be class 2 Operation, so it only can obtain connections at a distance of ten meters, which if it is intended to be used in a library, that distance is far too small to be able to work. If we had planned on using this in a smaller location, it would be possible, but if this were the case, the item watcher wouldn't be as useful anyway.

Another module under consideration is the BC04 Bluetooth module. It is capable of UART interfacing, and is said to have low power consumption. Although it doesn't specify the power consumption, it has low power modes park, sniff, hold, and deep sleep, available. The size of this module is 27.5 x 14.5 x 2 mm, which is pretty small, and would be lightweight as well. The price of this one can also be found at much cheaper than many other Bluetooth modules. As it seems, this would satisfy all the listed requirements, however there is speculation as to whether it is actually a class 1 operation; there are a few different websites listing it, and some say it can only obtain a class 2 connection.

An option that is fairly common for Bluetooth implementation is the RN41 module. It is class 1 operational, which is small and lightweight. The dimensions of it are that of a postage stamp. It has very low power consumption, both in active mode as well as sniff mode. While it is Bluetooth version 2.1, it also has an enhanced data rate that allows for faster communication, and it is backwards compatible. Another convenience with this model is how easily it can be incorporated into a project; it supports multiple interface protocols and has a high-performance antenna. It also is capable of UART communication, as is needed for our processor.

Another consideration for our Bluetooth module is the BLE112. This module is also capable of connecting to our processor, as is an expectation. It also consumes little power for its needs. This one however is only a class 2, which wouldn't allow for the range we would need for our project. However it can be powered by a simple 3V battery, where the RN-41 needs at least 3.3V. This is about the only place it exceeds it though.

The WT41 module is a sophisticated Bluetooth module with UART capabilities, as well as having a long distance connection able to be established, for this specific option it can get up to 800 meters. This is much farther than the previous connection capabilities, as a typical class one can connect up to 100 meters. This model is also able to have a USB interface mode that probably won't be needed, but is there if it ends up being useful. One downfall is that this model is more expensive than the previous ones, but this is expected because of the large range.

	Connection ability	Class	Low Power Consumption	Price
LMX9830	UART	2 – 10m	Yes	$13.46
BC04-B	UART	?	Yes	$9.96
RN-41	UART	1 – 100m	Yes	$21.70
BLE112	UART	2 – 10m	Yes	$21.00
WT41	UART	1 – 800m	Not specified	$33.00

Table 3.13-1

Based on the different modules which were compared, it seems that the RN-41 is the best option for the price. While the BC04-B is much cheaper, there were some sources that said it was a class 2, and others that said it was class 1; this uncertainty makes it a bad choice. The WT-41 does have a much wider range for connection, but it is also more expensive than the RN-41. Also, we won't need as far of a range as the WT-41 is capable of.

3.14 Batteries

Possibly the most important subsystem within BroBot is the power system. Without power, not one subsystem would function, from the motors to the software running on the microcontrollers. We have decided to implement a rechargeable battery for this project. While this will cost more initially, it will save us money in the long run on disposable batteries. Most of our hardware requires an operating voltage of around 3.3V, but our battery needs to be slightly higher to account for fluctuations. We are implementing a voltage regulator to limit this voltage where applicable. The only subsystem not powered by this battery is the motors powering the chassis. The motors come with their own AA batteries to use. In this section, we will examine each kind of battery to determine which is most appropriate for our use in the rest of BroBot.

3.14.1 Lithium-Ion

Lithium-Ion is the most commonly-used battery in cell phones and other electronics. They are rechargeable batteries, unlike normal lithium batteries that are not. Lithium-Ion batteries can be made several ways, such as Lithium-Cobalt-Oxide batteries that provide high energy but have inherent safety risks, Lithium-Iron-Phosphate, Lithium-Manganese-Oxide, and Lithium-Nickel-Manganese-Cobalt-Oxide batteries that provide longer life and safety but provide less power per unit area. These batteries function by the transfer of lithium ions from the anode to the cathode. During charging, this process is reversed.

Lithium-Ion batteries are more dangerous than other kinds of batteries, because they are kept pressurized and contain a flammable electrolyte. While safety features are built in to these kinds of batteries, many accidents have occurred from their use.

Lithium-Ion batteries have a decaying battery life. It lasts the longest when new, and decreases with age of use until it is unusable. Their battery life is also dependent on temperature. The hotter the battery, the quicker they die. This should not be a problem, because our intended use is within an air-conditioned library.

Lithium-Ion batteries are especially advantageous due to their comparatively high energy densities. The energy density of a lithium ion battery ranges between 250-360 W*h/L. This will allow us to get the voltage we need with the minimum size. Since we only have limited chassis space, shrinking the battery is advantageous for us. If it is small enough, we can put the battery on the PCB itself if there is room. If not, we can keep the battery external.

3.14.2 Nickel Metal Hydride

Another kind of battery is called Nickel Metal Hydride, or NiMH. It is also rechargeable, utilizing the positive electrodes of nickel oxide hydroxide and a hydrogen alloy as the negative electrodes. It is very similar to the now-obsolete nickel cadmium batteries. A nickel metal hydride battery has two to three times the capacity of a nickel cadmium, and has an energy density approaching the level of lithium ion. Specifically, nickel metal hydride has an energy density of 300 W*h/L. NiMH batteries are also cheaper than lithium ion batteries, saving us some money on our budget.

The biggest weakness of nickel metal hydride batteries is its fast rate of discharge. Nickel metal hydride batteries typically lose 20% of their charge on the first day, and 4% per day after that. There exists a low self-discharge variant of the NiMH, but this variety comes at the cost of 20% of its total capacity.

Overcharge can also cause damage to the battery. Also, NiMH batteries produce a lot of heat while charging, which can be a fire hazard. Overall though, NiMH batteries are safer to use than lithium ion batteries.

To safely charge a NiMH battery, a very low current is required. This causes the battery to take longer to charge. Also, charging the battery for longer than a recommended 10-20 hours can damage the battery. This should not be a problem, because our battery would be charged all night, every night, so overcharging will not be a risk. Also, the battery will have time to slowly charge up.

NiMH batteries typically provide about 1.25V per cell, so we would require at least four cells to meet our power needs. This could occupy more space than the equivalent voltage from lithium ion batteries. If we implement NiMH batteries, we will be forced to spare more room on the chassis for the power system, and will definitely not be able to fit the batteries on the PCB along with our processors.

3.14.3 Nickel Cadmium

Nickel Cadmium, or NiCd batteries use nickel oxide hydroxide as the positive electrodes just like NiMH batteries. However, NiCd batteries use cadmium as their negative electrodes. Their discharge voltage is only 1.2V, so we would need to use several as with NiMH batteries.

Many batteries do have a higher initial voltage than NiCd batteries. However, NiCd batteries see their voltage change very little over time, such that after a period their voltage will be higher than their competitors' which drops in potency with time.

Although NiCd batteries have lower energy densities than most other rechargeable batteries, they do offer some advantages. They are very durable, able to be used for long periods of time with few ill effects. They also can withstand more charge/discharge cycles than other kinds of batteries.

Unfortunately, NiCd batteries are fairly toxic and cannot compete with NiMH batteries in most categories. Although their self-discharge rate is lower, they have smaller energy densities, are more toxic, and are more expensive. For this reason, they cannot compete with NiMH batteries for our project.

3.14.4 Lithium Ion Polymer

Lithium Ion Polymer batteries, or Li-Po, is another form of rechargeable battery. They consist of cells arranged in parallel, and their output is proportional to the amount of cells arranged as such. Typically, a Li-Po battery ranges between 4.23V when fully charged to 2.7V when discharged.

Li-Po batteries have several advantages. They are extremely thin and carry a very high energy density. They also come in many different sizes, allowing us to select the one that would fit our space requirements best. They are also very safe batteries compared to the other types, however overcharging them can still lead to risk of explosion.

The major downside to Li-Po batteries is their need to be regulated. During use, as soon as each cell's voltage falls below 3V, they need to be removed and recharged, or they risk being permanently damaged and unable to hold as high charges in the future. Additionally, specific chargers are required for this kind of battery or they run the risk of catching fire, exploding, or both due to the arrangement of the cells.

3.14.5 Battery Conclusion

We have decided to go with a Tenergy Lithium Ion 7.4V 5200mAh battery pack. This battery comes with onboard protection to prevent overcharging or over-discharging. When the charge gets too high or too low, the battery is cut off to protect itself. This eliminates a major risk of explosion inherent in lithium-ion batteries. The voltage is high enough to power all of our systems, while low enough to not be wasteful. 5200mAh is plenty to ensure BroBot can go a long time before needing a recharge. It is very light and has a very high energy density. Relevant statistics are shown in table 3.14.5-1.

Voltage	7.2V (8.4V peak)
Capacity	5200mAh
Dimensions	135mm (L) x 37mm (W) x 21.5mm (H)
Weight	6.4oz
Cut-off Voltage	6V

Table 3.14.5-1

This battery will need to be purchased with an adapter to charge it with. Recommended for use with this battery is the Tenergy TLP-2000 Smart charger. This charger provides a constant current of 500mA when charging. It works with all 100-240V AC outlets. It automatically stops charging when the battery is fully

charged to prevent damage to the battery. This charger works for several batteries with a switch to select the voltage needed. For our battery, the charging voltage should be set at 7.4V.

4.0 Project Hardware and Software Design Details

4.1 Initial Design Architectures and Related Diagrams

Figure 4.1-1 shows the control flow for the item watching program. When the program begins executing, it saves a picture of the items it is watching. It then takes a new picture to compare the two. This picture is then compared against the initial picture. If it is similar, the cycle repeats. If it is not, the alarm is triggered and a picture and timestamp are sent to the user of the app. From the app, the alarm can be disabled. This will set BroBot back into the watching state where the cycle is continued. Not shown in this diagram is the occasional update of the initial picture. To prevent long-term changes like shadows from falsely triggering the alarm, the initial picture will be occasionally updated as well.

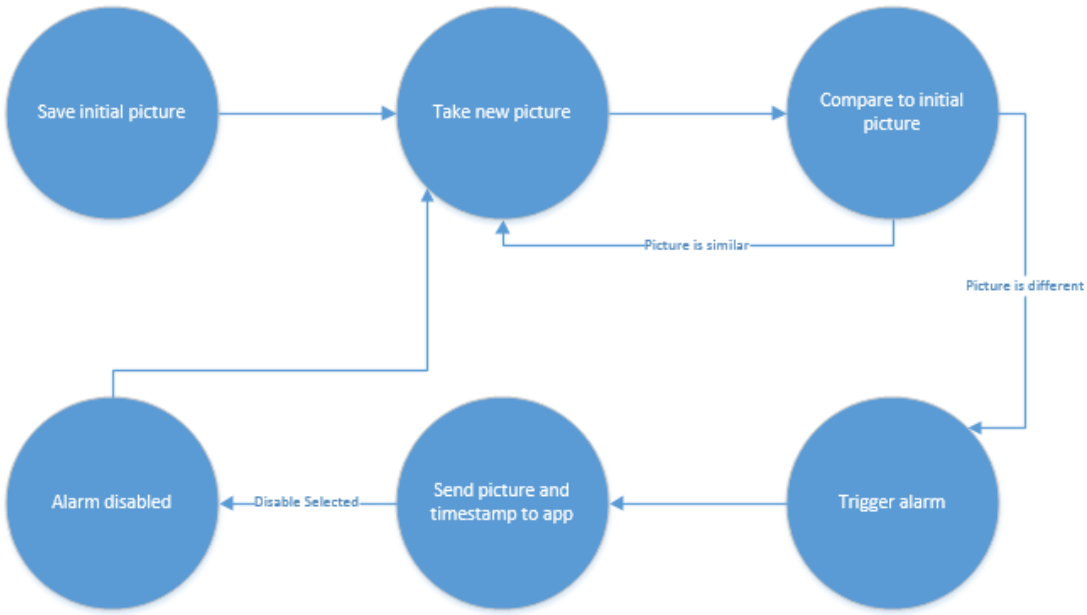


Figure 1.1-1

4.2 Item Watcher

4.2.1 Hardware Configuration

Diagram 4.2.1-1 shows the overall communications for the hardware side of the item watcher subsystem. The microcontroller in the middle of the system has control over all of the data flow. It will act as the master for all I2C communications for the subsystem. It will handle all of the data flow from the camera to any external thingy, and it will also be in control of the control lines of the camera. The microcontroller will also inform the navigation system of its duties, ie where to go. The microcontroller will go into a low power mode during navigation and will turn on when the navigation subsystem has completed its task. Also in this implementation will be 4 test LEDs, this are for the testing phase of the prototyping and also will be used for explaining different errors that the microcontroller could run into, for example too large of an image coming into the system, or if the external memory is full.

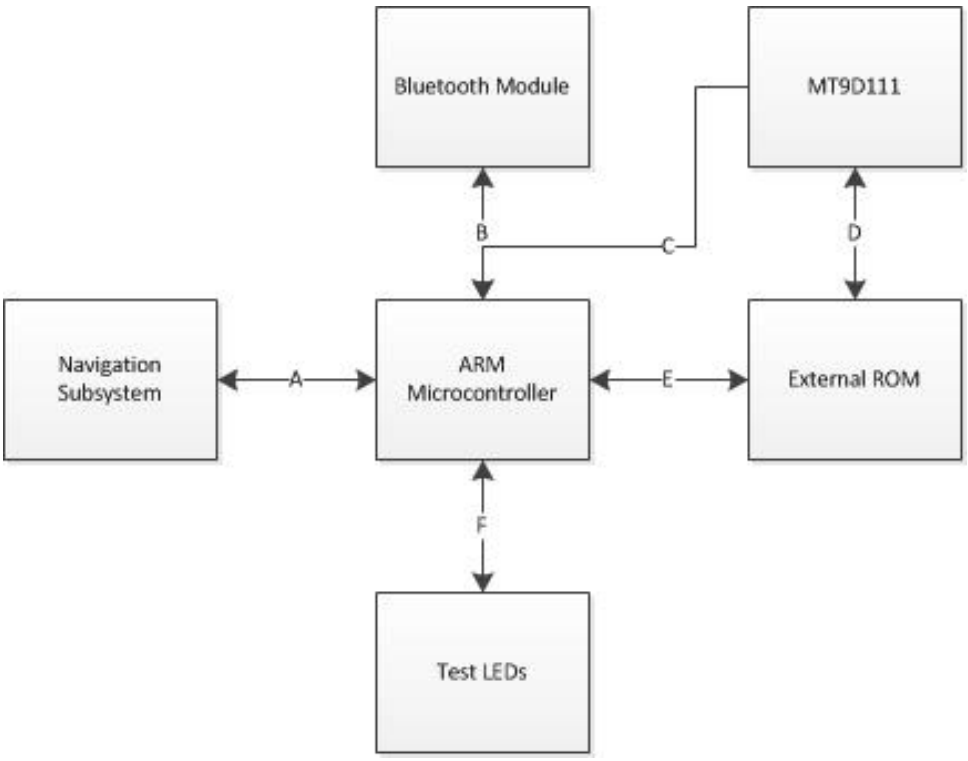


Diagram 4.2.1-1

Diagram 4.2.1-1 shows the different lines of communication that will be implemented for BroBot. The line A will be done using I2C with the navigation subsystem being a slave to the ARM microcontroller, which will act as the master. Line B and C in diagram one will also be I2C, but these lines will be on a different I2C port than line A, this is because these peripherals will be used during the

item watcher subsystem. This is so that the item watching program can be written only using one I2C port and will not have to worry about problems with the navigation subsystem. The information coming from the navigation subsystem will be small indications of problems or completion of the task. Line D and E are going to be the data lines coming from the ARM microcontroller. These lines will count for the most amount of lines coming from the microcontroller. Also for line E in Diagram 1 there will be the address lines that the external memory needs to know where to put the data.

Line F in diagram 1 will be GPIO pins coming from the ARM microcontroller. There will be 4 different LEDs, one will show power, and the other three will be used for debugging. With 3 LEDs come 8 different error calls that can be sent to the user. One of the LEDs will be used to show power to the microcontroller. The overall use of these test LEDs are for testing purposes and are subject to change since they might not be needed in the final prototype.

4.2.2 Camera

The MT9D111 camera module by micron is going to be used in this project. The MT9D111 gives a lot of flexibility when it comes to the compression of the image files. Since this is such a big deal for an embedded system with a small amount of memory this camera was chosen. Figure 4.2.2-1 shows the breakout of the module that the camera is on. The camera has 20 different pins that need to be used, though not all of the lines are going to the microcontroller. 10 wires will go from the camera to the microcontroller, these are the 8 parallel data lines and the 2 serial control lines. The camera will need only need one line of voltage to power it at 3V. The other pins on the camera module are used to control a mechanical shutter and an external flash. These pins will be set to ground since they will not be used.

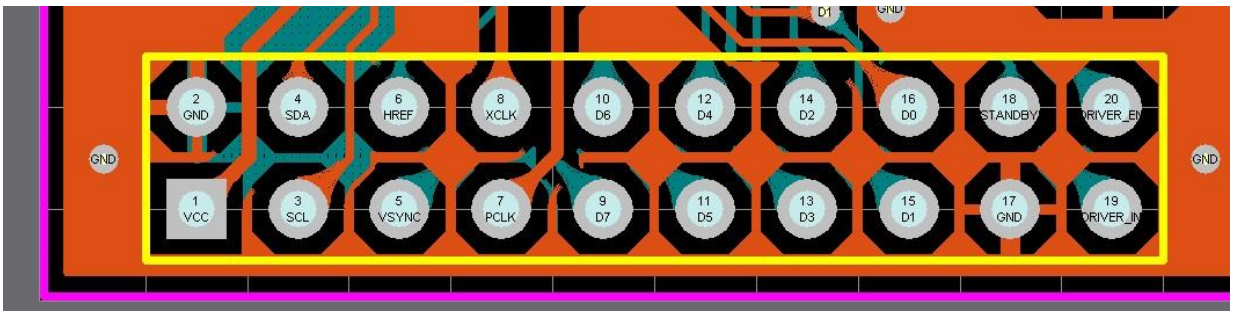


Figure 4.2.2-1 with permission pending

4.2.2.1 Initializing the Camera Module

To start the camera module a sequence of events must occur. Power up the power supplies, provide an input clock, and perform a hardware reset. To perform a hardware rest the sequence of events below must be followed. This is done when all supplies are stable.

1. Give the module an input clock
2. Make RESET# low for at least 1us
3. Make RESET# high (while input clock is running)
4. Wait 24 clock cycles before using the two-wire serial interface

The power for the three lines are as follows, the analog voltage is at 2.8V for the best image performance, the digital voltage is 1.7-1.9V while the I/O voltage is 1.7-3.1V. Before the camera module can be used 5 different subsystems need to be configured and enabled. This is done via the serial interface. The subsystems are:

- PLL
- Pad slew rate
- Preview mode
- Auto focus
- Capture mode

Each subsystem has their own needs when it comes to setting them up. The PLL is to help the serial communication work with faster frequencies and will be set up for 10Mhz, which is the standard setup configuration for the PLL. To configure and enable the PLL you first need to program the frequency settings, then power up the PLL by setting R0x65:0[14]:0. After these first two steps you then have to wait for the PLL to settle, which takes about 200us. Next and finally you will need to turn off the PLL bypass by changing R0x65:2[15]=0. It must be noted that while PLL isn't enable the two wire serial interface will be limited in speed. Once it is enabled the communication speed can be increased.

Next to configure is the pad slew rate. This will be configured to the default setting since some tests will need to be run to see if the slew rate from the data or the serial interface is causing a problem. Next is the preview mode, which defaults to 800x600 and up to 30 fps. This mode will not be used for this project so everything will be left in default. Next the auto focus will be configured, for our project this will be set to snapshot mode.

In snapshot mode the camera will autofocus each time a user commands to do so. The autofocus will occur using its own autofocus algorithm, then it will take a picture and wait for the user to send another command. Finally the capture mode will be configured for the different features that are needed in this project.

4.2.2.2 2-Wire Serial Control Line

From the serial control line the camera and encoding can be set up to whatever is needed for the given project. It has full control over what type of image or video will be sent out, the resolution, and the quality factor if the image stream is compressed. The control line will also tell the camera module when to send a

picture. The control line can send information on the picture like its size, which is be useful for the microcontroller to know if the images are too big. From the control line different registers can be manipulated or checked upon, this is so to give the user complete control over the camera module. All of these registers and how to access them are in the camera's data sheet.

The control serial lines have a specific protocol for interfacing, which might cause some problems when first implementation. The protocol works a lot like I2C interfacing but with some minor but important differences. The two lines of communication are SDATA and SCLK, one line being the data line and the other being the clock line. SDATA is pulled up by the camera module and can be pulled down by the master or the module. It works on a master/slave model similar to I2C. The protocol for the bits are as follows:

- A start bit
- The slave 8-bit address
- A(an) (no) acknowledge bit
- An 8-bit message
- A Stop bit(or another acknowledge bit and continuation of the stream of data)

With all of these include that makes one stream of information to the camera module 19 bits long, meaning there is a minimum of 19 clock cycles that need to be inputted at a minimum into the camera module.

A typical read or write sequence follows this type of structure:

1. Master sends a start bit with a 8 bit slave device address, of which the last bit of the address indicates either a read or a write, with “0” being a write and a “1” being a read
2. To acknowledge the address the salve device sends an acknowledge bit to the master
3. If the request is a write than the master will send a 8 bit register address to the camera module
4. Once again the slave sends an acknowledge bit to the master, which tells the master that the address was received
5. Then the master will transfer the data 8 bits at a time, the slave sends an acknowledge bit after each 8 bits

The registers inside the camera module are 16-bits long, therefore it takes 2 data cycles to write to one register. After 16 bits have been inputted the microcontroller on the camera module automatically increments the register pointer and moves on to the next register. This action doesn’t stop until the master sends a stop bit after the data.

According to the implemented protocol inside the camera module a start bit is defined as a high to low transition on the data line while the clock is high. The stop bit is defined as a low to high transition on the data line while the clock line is high. For one bit of data to transfer the data line needs to be stable during the high period of the clock line. The data line can only change when the serial clock is low. For the acknowledge bit the receiver will pull the line low when it receives the acknowledge clock pulse. A no-acknowledge bit is used to stop a read sequence and is defined by when the data line is high from the receiver during the acknowledge clock pulse. Figure 4.2.2.2-1 shows a write timing to R0X09:0—Value 0x0284.

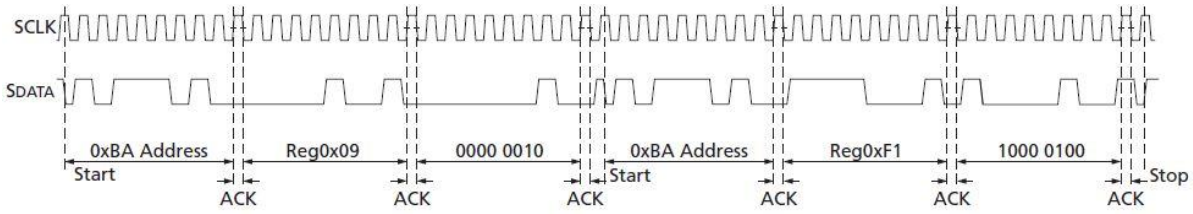


Figure 4.2.2.2-1 with permission pending

This special serial communication will be implemented using a bit banging system with 2 GPIO pins acting as the data line and the clock line. Though this will take more time to implement through programming we will have complete control over this line of communication. Another consideration is the timing of the SCLK, a time of 1MHz will be implemented first and tested out.

By default the sensor serial bus responds to addresses 0xBA and 0xBB, when SADDR is set the sensor will respond to addresses 0x90 and 0x91. Since it doesn't matter which slave address we use we will just ground this pin so that the module will react to the addresses 0xBA and 0xBB.

Whenever the camera is not in use but overall system is powered, ie navigation, the camera will be put in standby mode, which is its lowest power consumption state. The standby pin shown in figure 2 controls weather or not the camera is in standby mode, but also a command line must be sent to a particular register on the camera module's microcontroller.

4.2.2.3 Still Mode Configuration

The resolution that will be used will be at 800 x 600 resolution, this is to ensure that the picture will be big enough to include everything in front of the camera but not too large that the system will not be able to work with it. The jpeg configuration will be at 4:2:0 chroma subsampling, this is the lowest setting that the camera has for the Chroma sampling, another consideration is to output greyscale images. The initial quality factor will be 20 from the camera module.

This configuration could be subject to change during the testing part of the prototyping process. Since the camera module has the ability to change the quality factor of the image coming in there could be a lot of change in this design. All the other settings will be put to default, this is mostly camera settings like auto focus or optical zoom, since they will not be needed for this project.

Since the manual focus will not be used for this project the camera will be used in the snapshot mode. In this mode the module will auto focus with a command from the microcontroller. Once the camera auto focus is finished then a picture will be taken. To be able to take and output pictures the camera will also be in snapshot mode, this will take full use of the modules ability to capture and encode a taken image. The auto exposure feature needs to be set in a specific mode as well. The mode that will be used in this project will be the scene evaluative algorithm mode, since this will adjust the exposure so to ensure the best quality of image coming into the microcontroller.

To control this configuration some variables on the camera module need to be adjusted. All of these variable can be either read or written, format, a uchar, controls the color scheme, with 1 being the one that is desired (4:2:0). The config variable is used for overall configuration and handshaking. This variable will be either 0x78 when the microcontroller is ready for a picture or 0x70 when the microcontroller isn't ready for a picture. The final registers that are of use for this project are the datalengthMSB and the datalengthLSBs, these show the previous frame's data length. This variable will be used during testing to see how big the images are for different settings.

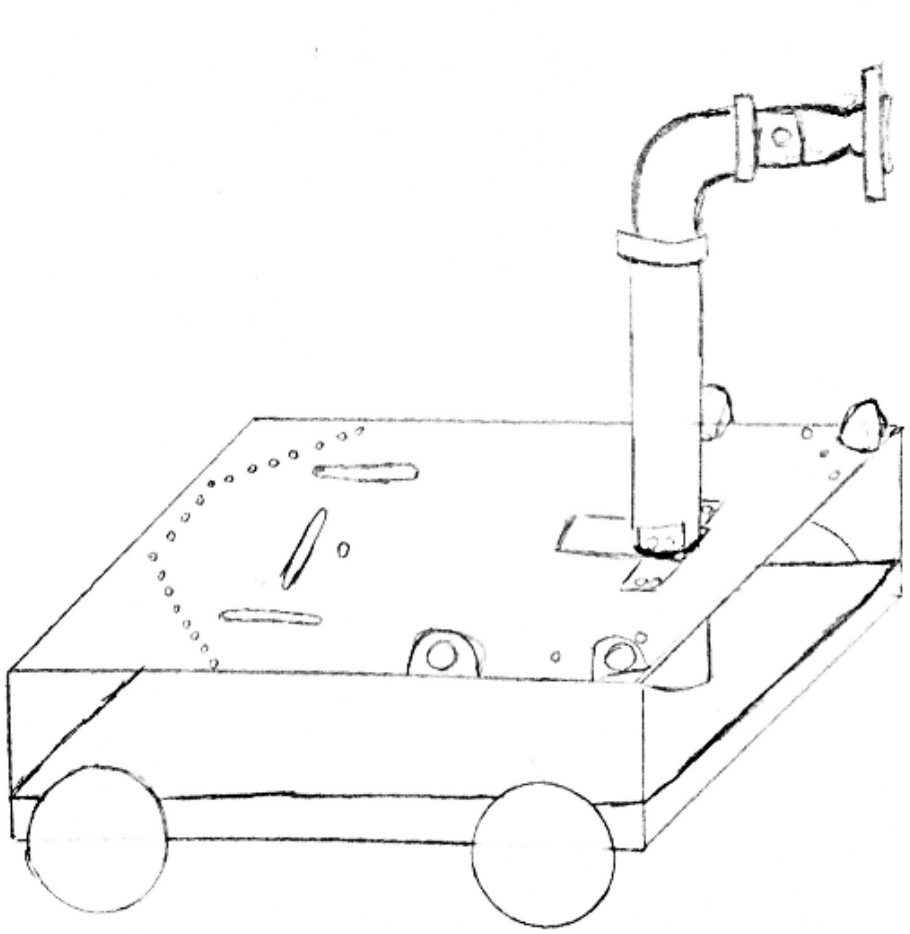


Figure 4.2.2.3-1

The camera will be mounted onto the chassis of the robot so that it will be easy to put BroBot on a chair and have the camera be able to get a good picture of what the user wants. An arm will be used to hold the camera up if BroBot is needed to be put on a chair, this is shown in figure 4.2.2.3-1. Also on top of the arm will be a piece that will be able to swivel the camera so brobot's vision can be manually adjusted to include everything that the user will want to see.

4.2.2.4 Output data configuration:

For the MT9D111 the jpeg data is output with an 8 bit parallel bus, dout0-dout7, frame_valid, Line_valid, and pixclk lines. The camera module will output the pictures in spoof mode. This mode is to make the output look like a normal video stream, with a set size of packages for the picture. This mode is going to be used due to how the microcontroller deals with taking in the information. In this mode the Line Valid line acts like the Vsync line in the microcontroller and the

data valid line will act like the hsync line in the microcontroller. Figure 4.2.2.4-1 shows the timing for the spoofing configuration.

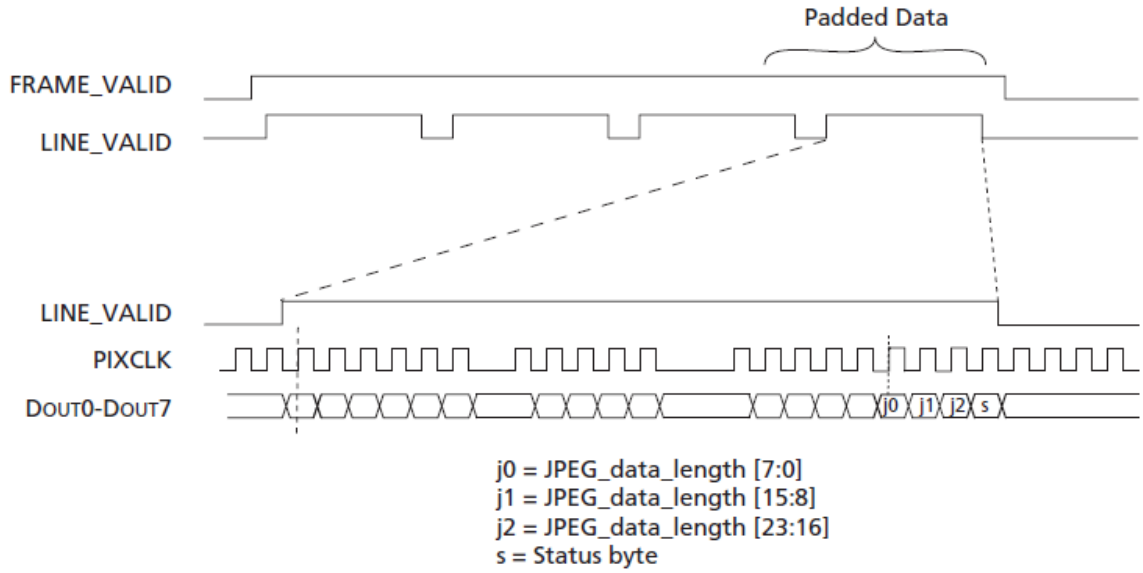


Figure 4.2.2.4-1

Every jpeg image that is streamed has padded data at the end of the stream as shown in figure 10. When line valid is high that indicates a correct package. This timing is not what will be exactly implemented, the pixclk line is gated. In the real implementation the pixclk line will not be gated to ensure nothing weird will occur.

The microcontroller will take all the data and store it on the microcontroller so it can do the needed calculations for the item watching process. Some variables are used by the microcontroller on the module to hold important values, including the size of the picture and the configuration of the JPEG encoder.

4.2.3 Item Watching Program

OpenCV, upon which we plan to use some functionality, is usable in conjunction with C++, C, Python, and Java. Of these, C is the most adaptable and malleable for use on low memory platforms such as the microprocessor we plan to run it on. Moreover, members of the team already have some experience in image analysis using C, so it will be an easier extension to utilize OpenCV with more familiar techniques. On the downside, using C is much more complex than using Java. Memory management must be dealt with, and pointers complicate the code.

Using Java would be much easier to code, since OpenCV already supports it, and the language provides automatic garbage collection and automatic pointers in the form of object references. Unfortunately, its ease of use comes with a

price. Java is not as flexible as C, and we have a very limited amount of space on the microprocessor. In addition to the code, we need to be able to store two pictures taken by the camera at a resolution high enough to perform analysis on. This feat may not be possible with just the on board memory, so we may incorporate some SRAM to help with the load. Even with this help, we would like to exert the most control over our memory use, so we will be using C.

This software will be running on an STM processor, so we need to ensure compatibility between the software and the controller. Using a Bluetooth connection, we will communicate from the item watching program to the app running on the user's phone. Information will not be constantly transmitted, it is only necessary to send information upon initialization and if an object is stolen from BroBot's field of vision. Although Bluetooth has a max range of about 100 meters, the audible alarm functionality will continue to safeguard items even while the user is outside of this range.

The program works by storing an initial picture of the items to watch. This picture is constant and unchanging and saved as a matrix of pixels, where higher numbers correspond to a higher brightness. After some interval of time, a new picture will be taken. This picture will be compared to the original one by comparing them pixel-by-pixel and possibly using third party tools such as background subtraction, detailed in the API section. The magnitude of the brightness of this picture will be summed up by taking the square root of the square of the difference of the two. If this magnitude exceeds a threshold, found by experimentation, it means it is different enough to be considered an entirely different picture. This could be due to fringe cases like a change of brightness over time or something new being added into the picture, but will most often be the removal of an object. When this value crosses the threshold, it will trigger the alarm functionality on BroBot.

Although the pictures taken over time should be similar to the initial picture, subtle changes can occur that will make this not the case. For example, a change in brightness or shadows can corrupt the picture. These changes usually occur gradually. We want to be on guard against sudden changes, and be able to successfully ignore subtle changes that should not trigger the alarm. To accomplish this, we will update the initial picture, against which the subsequent pictures are measured, every minute or so. If we did this update quickly, every second or two, it would allow someone to slowly move an object out of frame without being detected. If we did this update too slowly, changes in shadows and lighting could occur before the initial picture updated to take this into account, falsely triggering the alarm. This gradual change of the initial picture needs to be in that in-between range where it is slow enough to prevent actual sudden changes to the picture but fast enough to change before the shadows or lighting changes caused the picture difference to cross the threshold.

The alarm being triggered will do a few things. First of all, it will set off an audible alarm to alert people nearby that something has been taken, and they can intervene. It will also send the current picture to the user's app for verification, along with a timestamp. If the user sees that something is missing, they can rush back to defend his or her things. If it was a false alarm, the user can press a button to send BroBot back into watch mode.

4.2.4 ARM Microcontroller

We are using the STM32F407VGT6 microcontroller due to its ability to easily take in 8 bit parallel data and its large amount of memory space. The microcontroller will be in a 100 pin package, the pin out is shown in figure 6. To communicate with the different peripherals 2 I2C ports will be used. The microcontroller will act as the master in the I2C system. By default the microcontroller is in slave mode, so this will need to be changed in the programming.

4.2.4.1 2-Wire Serial Communication

The M9D111 cannot be implemented using the I2C system since it sends 8 bits of data information and I2C only supports 7 or 9. The USART features can't be used because they have to send an end bit and a starting bit, which would be too mean the information would overflow to the serial line for the camera. Also the UART system is a 3 wire system and a selector would also need to be implemented into the system, further complicating things. Because of this the communication line will need to be implemented using 2 GPIO pins on the same port. They need to be on the same port due to the fact that they need to change simultaneously.

The fastest the GPIO pins can toggle is every two clock cycles, with a clock cycle of 168MHz that means the fastest rate this can be implemented is at 89MHz, which would be overkill for the camera system. The program for the control of the camera will need to throw out clock cycles so that the camera module can receive the information. The camera module interface will initially be run at 1MHz, though this could be easily changed in the programming during the testing of the camera module.

The two serial lines will initially be high and therefore will need to be pulled down by the master system, which in our implementation is the ARM microcontroller. The line will be at 5 volts for high and ground for a low voltage. This is the maximum that the ARM microcontroller can output, therefore some other resisters will need to be implemented so that this system will run smoothly. There are pull up and pull down resistors on each of the GPIOs on the microcontroller which eliminates the need for added extra resistors, which saves on PCB space. The structure of the five-volt tolerant I/O port bit is shown in figure 4.2.4.1-1.

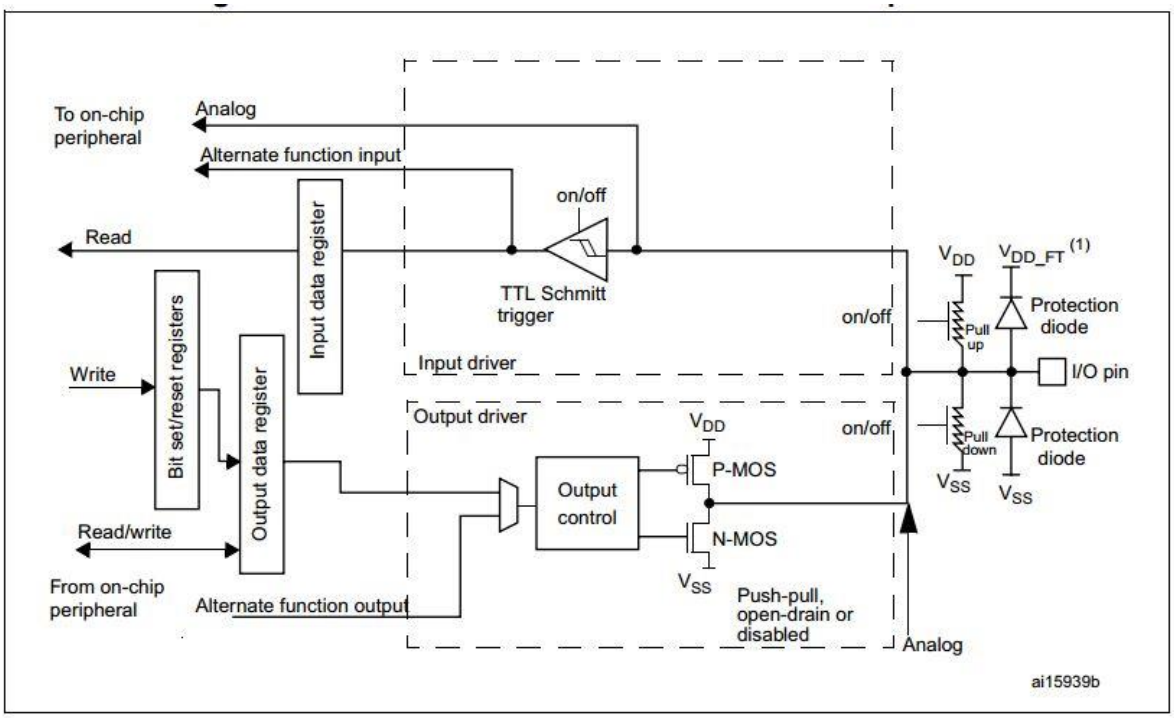


Figure 4.2.4.1-1

The configuration of the ports for output will be open drain pull down, since the line will be driven high by the camera module. The input will be pull down, for the same reason that the output is open drain. For an input a pull down resistor will be activated. This is to ensure that the internal hardware of the microcontroller will not be damaged. Since the microcontroller has many GPIO pins, the pins that will be used will not be shared with another system. For example there will need to be I2C communication between the navigation system and the microcontroller, the pins for the I2C communication will not be used for the bit banging operation of the camera module.

4.2.4.2 Digital Camera Interface (DCMI)

The digital camera interface built into the microcontroller is designed for easy integration with camera modules and CMOS sensors. This feature was the main reason for choosing this ARM microcontroller. The DCMI can have 8-14 bit parallel synchronous interfacing, has a continuous or snapshot mode, and supports compressed JPEG data. This functionality assumes that all the pre-processing is done by the camera module, like resizing and cropping. There are 17 pins associated with the DCMI, 14 are for the data lines, and the other three are HSYNC, VSYNC, and PIXCLK. The two clock domains are the PIXCLK and HCLK, PIXCLK is a derivative of HCLK and the period of PIXCLK must be higher

than 2.5 HCLK periods. For JPEG image reception the JPEG bit must be set. This bit is bit 3 of DCMI_CR register. Figure 4.2.4.2-1 shows the DCMI block diagram, showing how the synchronous interfacing works.

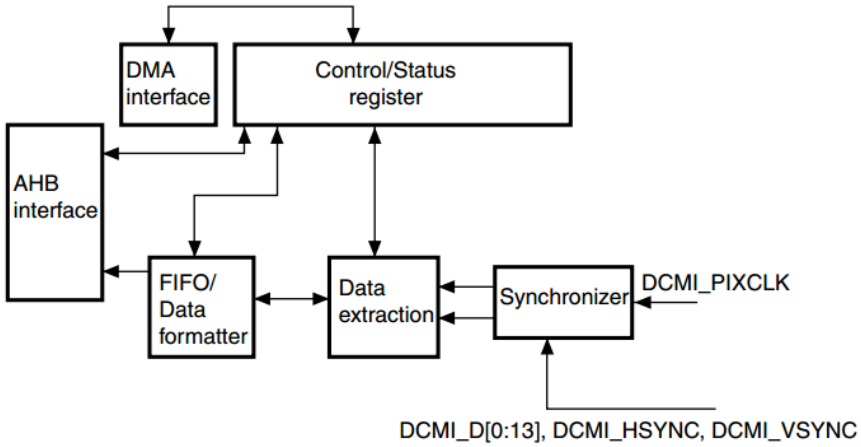


Figure 4.2.4.2-1 with permission pending

To capture the picture that is coming in on the DMA interface the CAPTURE bit in the DCMI_CR register must be set. Since the camera sends out information in 8 bits the 8 bit option will be used in the microcontroller. To go into this mode EDM[1:0] in DCMI_CR need to be 00, this will capture D[0:7]. Since the data will be placed in 32-bit words the first byte will be in the LSB position in the 32 bit word, while the 4th byte will be in the MSB position.

The synchronization of the data from the camera module to the microcontroller is one of the most important systems that need to work well in the system. Since the microcontroller has a lot more restrictions when it comes to interfacing with JPEG image information, so the camera module will be changed from default to meet with the needed settings. For the JPEG input mode on the microcontroller only the hardware synchronization mode will available. For this mode VSYNC on the microcontroller shows the start/end of an image, while the HSYNC is used as a data valid signal. Figure 4.2.4.2-2 shows what the microcontroller needs, it also is important to see that this figure and figure 10 look the same, this means that integration will be possible.

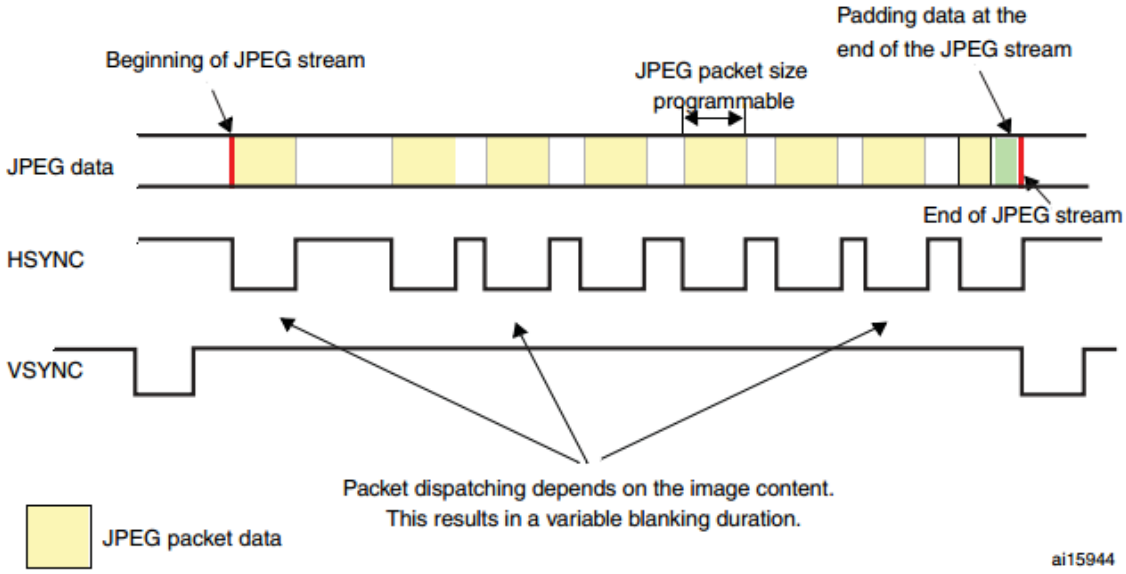


Figure 4.2.4.2-2 with permission pending.

The snapshot capture mode will be used in the DCMI. Once the capture bit is set in the DCMI_CR the system will hold until the start of a frame. After the first frame is sent the capture bit is cleared in DCMI_CR. Also the IT_FRAME interrupt will be generated, if it is enabled. For our system we will enable that interrupt, this is so it will be easy to know when a picture is done coming in and then it can be worked on. If there is an overrun then picture will be lost, but the capture bit will be cleared. Due to the way JPEG is encoded and compressed there are no limits of the input size for the image.

The DCMI is controlled by register DCMI_CR on the microcontroller. Before enabling the DCMI the entire register should be configured. Therefore when the interface will be enabled only one bit will change on the register. Before enabling the number that will be inputted is 0x00EA, though some of those bits need to be changed to zero and some do not need to be changed. Also some interrupts will be enabled in the DCMI, therefore the DCMI interrupt enable register (DCMI_IER) needs to be adjusted for such. The overrun interrupt and the capture complete interrupt will be enabled, these two interrupts are enabled on the last two bits of DCMI_IER, and therefore the number inputted will be 0x0003.

4.2.4.3 I2C

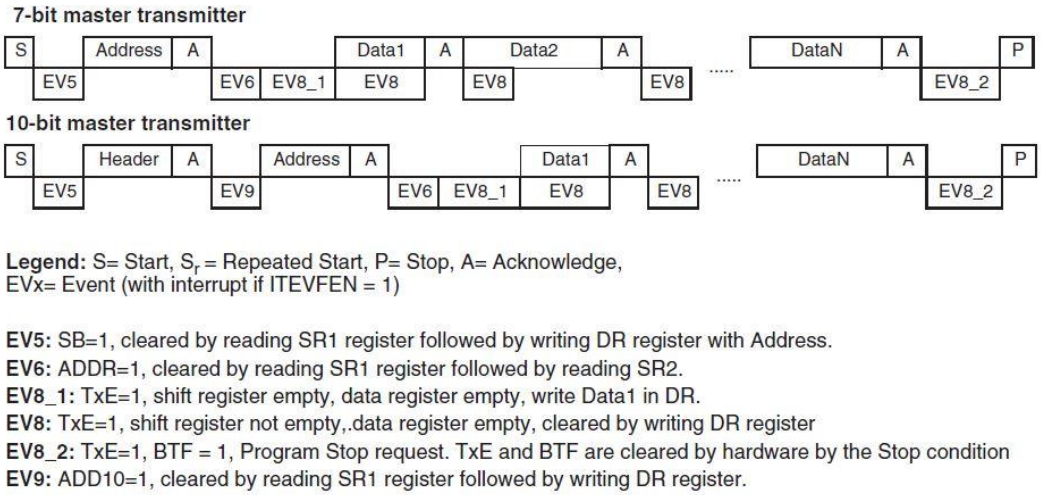
To communicate with the navigation subsystem the serial protocol I2C will be implemented. The ARM microcontroller will act as the master in the system while the microcontroller that deals with navigation will be the slave. The system on the microcontroller can support the standard speed (up to 100kHz) and the fast modes which can be as high as 400kHz. For this project the standard up to 100kHz speed mode will be used, since the other microcontroller might not have this functionality and that would cause problems, also the speed at which this data is transferred doesn't need to be sent extremely fast.

Over this line simple communications will occur between the two microcontrollers. The information shared will be first that the navigation system needs to wake up, then it will send the destination through I2C. Since the final number of destinations will be a small number one 7 bit number will be fine to send over, since there is a combination of 127 locations. Since this communication between these two microcontrollers will be the only information going over the I2C bus then only 2 lines will be needed from the ARM microcontroller.

The microcontroller can be put into 4 different modes slave receiver/transmitter, and master receiver/transmitter. As previously stated the microcontroller will act as the master and therefore be only in master receiver/transmitter modes. To initiate the system the microcontroller will be in the master receiver mode. For master mode the following sequence is required:

- The peripheral input clock in I2C_CR2 will need to be programmed in order to produce the right timings.
- The clock control registers will need to be configured.
- The rise time registers need to be configured
- I2C_CR1 register will need to be program to enable the peripheral to start and act as the master
- Finally the start bit in the I2C_CR1 register will need to be set to generate a start condition.

Once the start bit is set the interface will start to generate a start condition. The sequence diagram for master transmission of the I2C system is shown in figure 4.2.4.3-1. The top part of the diagram shows what will be sent on the SDA line while the bottom part shows what happens within the hardware part of the system, for example a specific register is going to be cleared or whatnot.



Legend: S= Start, S_r = Repeated Start, P= Stop, A= Acknowledge,
EVx= Event (with interrupt if ITEVFEN = 1)

EV5: SB=1, cleared by reading SR1 register followed by writing DR register with Address.
EV6: ADDR=1, cleared by reading SR1 register followed by reading SR2.
EV8_1: TxE=1, shift register empty, data register empty, write Data1 in DR.
EV8: TxE=1, shift register not empty,.data register empty, cleared by writing DR register
EV8_2: TxE=1, BTF = 1, Program Stop request. TxE and BTF are cleared by hardware by the Stop condition
EV9: ADD10=1, cleared by reading SR1 register followed by writing DR register.

Figure 4.2.4.3-1 with permission pending.

Every event can throw an interrupt if the ITEVFEN bit is set. If the system doesn't have enough time to get the data onto the buffer then the SCL will be stretched low. Once the last byte of information is on the DR register then the STOP bit needs to be set by the software, this will generate a stop condition. After the stop condition the interface automatically goes back to slave mode.

After the address transmission and the clearing of the ADDR by the hardware this interface goes into the master receive mode. Once in receive mode the interface can receive bytes from the SDA line, these bytes are put into the DR register, after each byte the interface will generate a specific sequence, first an acknowledge pulse, if that ACK bit is set, and the RxNE bit is set, this will trigger the interrupt if the ITEVFEN and ITBUFEN bits are high in the configure register.

When the last bye is received from the slave the master will send a NACK, when the slave receives this NACK it will lose control of the SCL and SDA lines, then the master can send a Stop/Restart condition. In order to create a NACK pulse after the last data byte is received the ACK bit must be cleared just after the second to last byte of data is read. To do the stop/restart condition the software needs to set the stop/start bit after reading the second to last data byte. Since we know exactly how big the amount of data that will be sent is we won't have too many problems implementing these types of implementation settings inside the software. Since we will only be sending one byte the ACK disable has to be made during the EV6 (shown in figure 12) then the stop condition will be made after EV6.

A couple different errors can occur for this interface, some of which can throw interrupts if the error occurs. A bus error happens when the interface sees an external STOP/START condition during a data or an address transfer. When this occurs the BERR bit is set and an interrupt would be generated. It must be noted that ITERREN bit must be set to have the interrupt occur. Another error would be

an acknowledge failure, this happens when the system detects a nonacknowledge bit, this is a good thing when you want to stop communication, this will set the AF bit and throw an interrupt, if ITERREN is set. If a transmitter receives a NACK then communication will need to be reset, if it's a slave that lines will be released by the hardware and if it is a master a stop/start condition needs to be generated by the software.

The I2C interface does have a programmable noise filter, this is to ensure the system will be within the correct protocol of the fast mode. Since the project will use the normal speed for the I2C interface this feature will more than likely not be used. If it turns out our I2C system doesn't work as intended then we might need to use some type of noise filter. Also our system will be on a PCB, which is to ensure there shouldn't be too much noise coming into the system.

The final component of the I2C system that needs to be designed will be what bits will be changed in the registers during the different phases of communication. The first register is the I2C controller register 1 (I2C_CR1). Bit 11 is the POS bit and will not be used for our project since it deals with 2 byte reception. Bit 10 is the acknowledge enable, it will send an acknowledge sequence after the next byte of data. Bit 8 and 9 are stop and start generations, which were discussed in the paragraphs above. The only other bit in this register that needs to be changed from a '0' to a '1' is the first bit, which is the peripheral enable bit. Also this bit must not be changed before the end of communication when we use it, since we will be using the microcontroller in master mode.

I2C control register 2 (I2C_CR2) also needs to be set during and for operation of the I2C interface. Bit 12 is what is used in master mode to allow the system to create a NACK on the last received data. Bit 10 enables the buffer interrupt which will be enabled during the testing of the I2C system, also bit 9 and 8 enable the interrupts. The final 6 bits are used for the peripheral clock frequency the range is 2Mhz to 42Mhz. For our system we will start with a smaller frequency of 10 Mhz (0b000110). The I2C data register holds the data that will be sent to the slave, the bits that are used to store the data are bits [7:0].

The last register of interest for our project in the I2C interface is the I2C status register 1 (I2C_SR1). This register holds information that will be integral to understanding problems that might occur in the system during testing. Bit 10 is the acknowledge failure, bit 9 is arbitration lost, bit 8 is a bus error, bit 2 is the byte transfer finished bit, and bit 1 is the address sent mode. All of these bits will need to be tested during operation and testing to make sure the system is working correctly and the communication is working as set up.

While the I2C interface is a very nice feature of this microcontroller there are a lot of options that need to be correctly lined up with the navigation microcontroller to ensure proper communication between the two systems. This part of the item watcher subsystem will need to have high priority for the testing of the systems.

4.2.4.4 Pin out of Microcontroller

Many different considerations must be made when the final pin out is determined for the microcontroller. Since the microcontroller has 100 pins to interface there should be no overlap of inputs. While not all 100 pins are used for interfacing, some are used for voltages and powering, a fair amount will be used by our system. Figure 4.2.4.4-1 shows the initial pin out of the microcontroller.

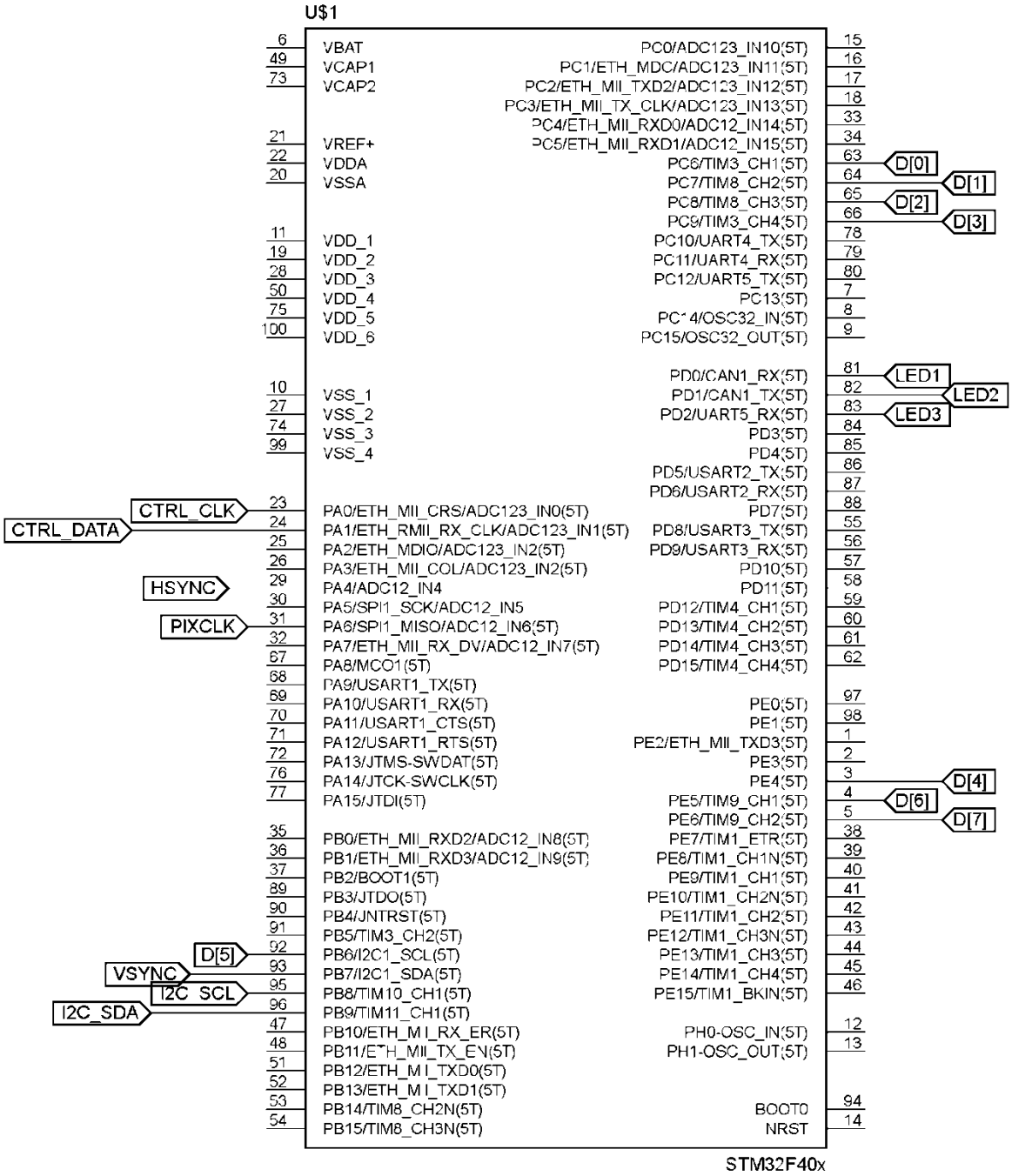


Figure 4.2.4.4-1 by Jacob Stewart

4.3 Navigation

4.3.1 MSP430G225

The MSP430G2553 will be the brains of the navigation system. This microcontroller will be in charge of the entire navigation operation, including sending and receiving information from the arm processor about the status of the navigation. It will need to be interfaced with 4 different sensors digital IR sensors and with the ARM Processor, which will be using the I2C serial communication protocol.

The microcontroller doesn't have many pins, only 20, so a big consideration that cannot be ignored is the amount of pins that will be used. This is why I2C will used for communication, even though it is harder to implement with 2 different microcontrollers. Some of GPIO ports might be used for Test LEDs that will help with debugging problems, and also help during the testing phase of the prototype cycle.

4.3.1.1 Operating Mode

During the navigation process the MSP430G2553 will be in Active mode, which takes the most power but has all the clocks active. Other than the active mode the microcontroller has five different low-power modes. All of them scale with the fifth mode having the least amount of features. An interrupt event can wake up the microcontroller from all of the different power modes, we will use this interrupt to wake up the Navigation system. When not in use the microcontroller will be in the lowest power state, low-power mode 4(LPM4).

In low power mode the microcontroller pulls around .1uA with a voltage of 3V, this mode is perfect for when the system is not in use. These modes are configured with the CPUOFF, OSCOFF, SCG0 and SCG1 bits that are in the status register of the microcontroller. Once the mode-control bits are changed then the operating mode immediately takes effect. In our code we will need to keep this in mind, so that no operations go on after we go into the low power mode.

To put the microcontroller in the fourth low power mode all of the bits that are mentioned above need to be to set to a value of 1. Also we will use the service interrupt to go out of the low power mode.

The microcontroller will be powered with a 3.3 V source on its DVCC pin that is shown in figure 4.3.1.1-1 below.

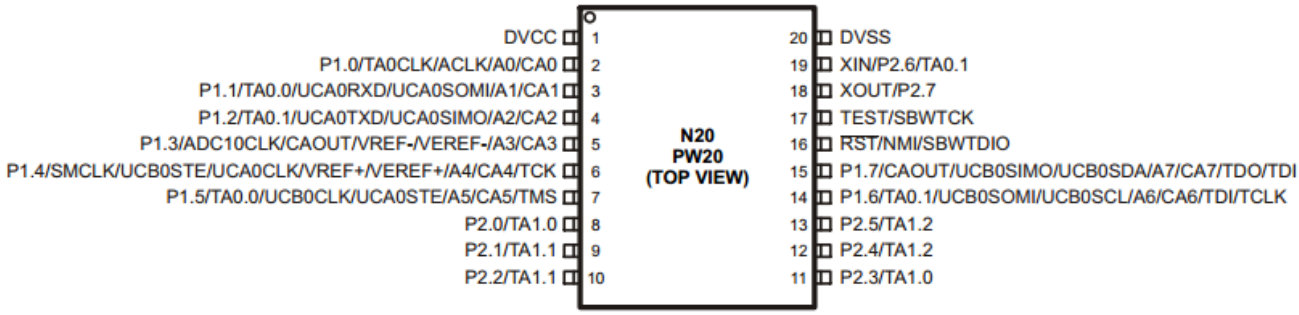


Figure 4.3.1.1-1 with permission pending

The microcontroller only has 20 pins, therefore a lot of consideration will need to be made to ensure that we won't use too many pins.

4.3.1.2 I2C

The msp430G2553 has a universal serial communication interface that can run in I2C mode. This interface will be used to communicate with the arm microcontroller that is in control of the entire system. This communication line will only send a 8 bit message, that will tell the MSP430 a couple of different operations. One of these operations is to tell it where to navigate to, also it will tell the MSP430 to go into sleep mode. Finally the MSP430 will tell the arm microcontroller when it has finished it job going to its destination.

To use the universal serial communication interface in I2C mode the USCI_Bx module will need to be used. The MSP430 will act as a slave in the system and will be using the 7 bit addressing mode. The system can support fast mode up to 400 kbps but we will use the 100 kbps standard mode for this communication. A nice feature of this interface is that it has slave operation in the LPM4, which is perfect for our implementation.

Figure 4.3.1.2-1 shows how the bus needs to be connected for the I2C setup.

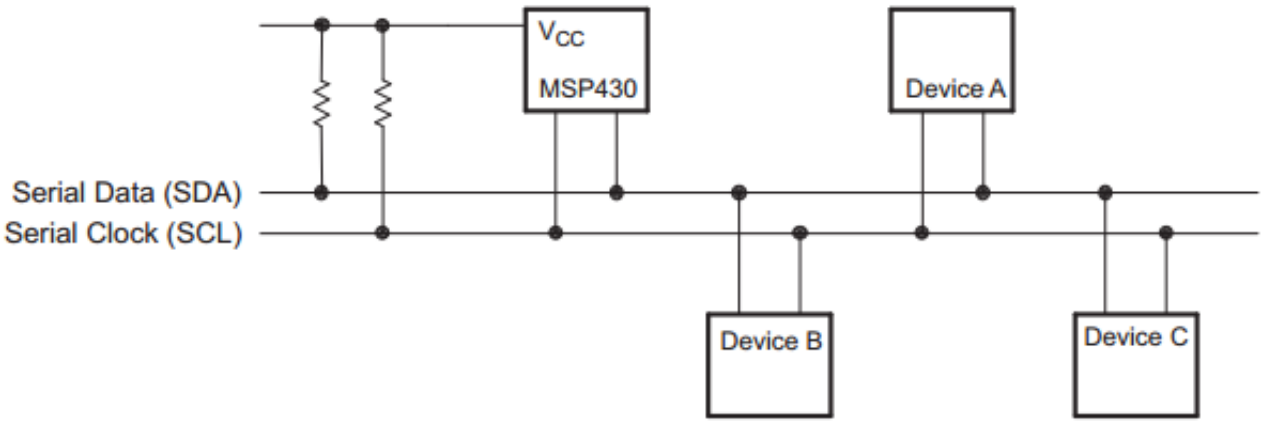


Figure 4.3.1.2-1 Permission pending.

This is the same setup that needs to be done with the ARM microcontroller, which is great! The USCI can be reset by a PUC or setting the UCSWRST bit. The mode is selected using the UCMODEx bits, for I2C mode these bits need to be set to 11. To stop using the USCI all you have to do is clear the UCSWRST bit. It should be noted that configuring the interface should be done when the UCSWRST is set. When this bit is set 6 different things happen:

- I2C communication stops
- SDA and SCL go to high impedance
- UCBxI2CSTAT[0:6] is cleared
- UCBxTXIE and UCBxRXIE are cleared
- UCBxTXIFG and UCBxRXIFG are cleared
- Everything else stays the same

For the 7 bit addressing, the first 7 bits are the slave address while the last bit is a R/W bit. To get the USCI module into slave mode the UCMODEx needs to be set to 11, UCSYNC =1, and the UCMST bit needs to be cleared. The module initially needs to be configured in the receiver mode, this is done by clearing the UCTR bit to receive the I2C address. Luckily the transmit and receive operations are done automatically depending on the R/W bit.

The slave address can be programmed into the MSP430. To determine the slave address we will flip a coin 7 times, with heads a 1 and tails 0. We have performed this operation and the slave address has been determined to be 0x90 for a read and 0x91 for a write. To program the address that number will be put in the UCBxI2COA register with the UCA10 bit 0.

After receiving a transfer call the UCBxTXBUF will be sent in the data section of the I2C. For a receive call the UCBxTXBUF will hold what is transferred. This register will hold what the arm microcontroller sends to the MSP430.

4.3.1.3 Motor Control

The motors will be controlled using a wheel encoder that is made specifically for the electric motors we are using. The encoder can increment the drive shaft by a set amount of degrees whenever a pulse is sent to in on the signal line. The encoder takes 3.3 V to power, and only has 3 pins. To have total control of the robot we will be using 4 motor encoders, one for each of the wheels. Each motor will be controlled using a different pin of the MSP430 microcontroller.

If we run into a problem of using too many pins then we could just use only two pins, one pin for the right wheels and the other pin would control the left wheels, this might actually simplify our project and will need less calibration, but would ultimately give us less overall control of the system.

4.3.1.4 PINOUT

Figure 4.3.1.3-1 shows the pinout that will be used for the MSP430 microcontroller. The Vdd will be at 3.3V while the Vss pin will be at the ground potential.

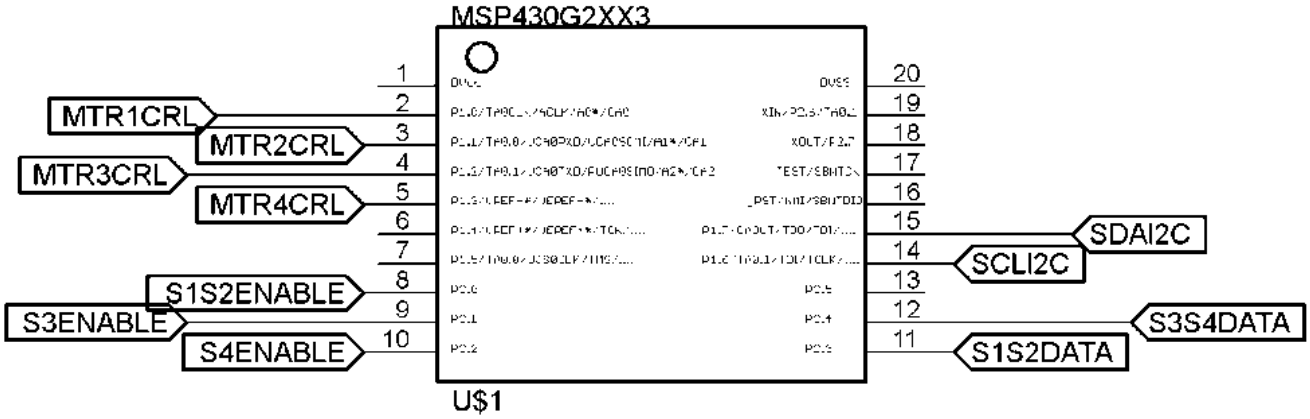


Figure 4.3.1.3-1

In figure 4.3.1.3-1 pins 1 and 20 are Vcc and Vss respectively.

4.3.2 Interfacing with the IR sensors

The MSP430 will be used to check the sensors during the movement of the robot. These sensors will be checked once every other second. During this time

it will sample a 10^{th} of a second or less to see if it will run into anything. Overall the implementation will use 5 different pins of the microcontroller.

To sample the data that is coming in an interrupt will be used to sample the GPIO output pin that we will need to look at. The pseudo code below will be used during the sampling of the sensors.

```
Enable sampling interrupt

While checking sensor{

If(counter>800){stop_motion=1;}

Else stop_motion=0;

}


Sampling interrupt{

If(Sensor1OUT == 1){counter++;}
```

Figure 4.3.2-1

This code will run during the time that the robot needs to sample the sensors. It will throw an interrupt 1000 times, when this interrupt happens it will check the sensor in increment the counter variable. While the sensor is checked the counter number will be looked at, and if it is over a certain threshold, for this example 800, it will stop operation.

4.3.3 Algorithm and Interrupts

The algorithm will be implemented on the microprocessor by creating a two dimensional array that will represent the library, with the dimensions being scaled down to two yards for the “length” of each segment in the array. After it is created, it will have specific locations in it marked to represent the different sections the library will be broken into. This scaling down will allow the microcontroller to better handle the program, as it will require less memory, and there will be a limited amount.

The infrared sensors will have a range of output voltages, depending on the amount of infrared light on the transistor components of them. When the values are converted to digital, as done by the microprocessor, they will be tested against some designated threshold value. Because the infrared values will vary for different colored surfaces, testing will be done to determine the threshold value needed for the correct detection. There will be an array of length four that will be used to represent the status of each of the four sensors. By taking the

read values and apply a threshold to them, we can then just check whether each sensor is on or off. Some situations will require that two sensors be turned on, such as the first two, and if this happens, when the robot is turning, the sensor on the side will need to be turned on. If one of the sensors on the side turns on, it will basically be meant that a turn can't be made in that direction. These will also be used to signify that the object that was being detected stays in view of the robot, just not in front of it.

As the robot travels from the initial location to the final destination, he will frequently check the states of the sensors, if there is ever something in front of him; he will try to turn in the direction that will keep him traveling towards destination. If he can't turn in that direction, he will turn in the opposite direction and try to turn back to the direction he was heading as soon as possible; this will be when the sensor on the correct side falls below the threshold. It should also be noted that the sensors on the sides and the front will have different threshold values, because of their need to detect objects from different lengths. To show what is meant by what the code is intended to do, Figure 4.3.5-1 displays the pseudo code which shall be used in writing the final code and helping to follow what will happen.

```
Set up front IR sensors to check status every 2 seconds
Set up side IR sensors to check when called

Main{
If(front sensors > threshold)
Call Directional_decision
If(the distance to end in x and y ==0)
END and stop moving
If (traveling in x){
If(Distance_traveled==(destination.x-position.x))
Call Directional_decision
}
If (traveling in y){
If(Distance_traveled==(destination.y-position.y))
Call Direction_decision
}


Directional_decision{
If(traveling in y){
If(destination.x>=position.x){
If(right_sensor<threshold)
Turn right
Else if (left_sensor < threshold)
Turn left
Else
Call Go_backwards
}
```

```
If(destination.x<position.x){
If(left_sensor<threshold)
Turn left
Else if (right_sensor < threshold)
Turn right
Else
Call Go_backwards
}

If(traveling in x){
If(destination.y>=position.y){
If(right_sensor<threshold)
Turn right
Else if (left_sensor < threshold)
Turn left
Else
Call Go_backwards
}

If(destination.y<position.y){
If(left_sensor<threshold)
Turn left
Else if (right_sensor < threshold)
Turn right
Else
Call Go_backwards
}
}

Go_backwards{
If(traveling in y){
If(destination.x>position.x){
While(right_sensor>threshold and left>sensor>threshold){
Reverse BroBot
}
Turn direction of off sensor
}
}
If(traveling in x){
If(destination.y>position.y){
While(right_sensor>threshold and left>sensor>threshold){
Reverse BroBot
}
Turn direction of off sensor
}
}
}

Distance_traveled{
Distance=speed*time_from_last_turn
}
```

Figure 4.3.3-1: Pseudo code for motion

4.3.4 Communication with the ARM processor

The chosen microprocessor, the msp430, is entirely in charge of the navigation of the robot. It should take care of the decision making, sensor reading, as well as controlling the motors to steer. While it will do all of this, it will also need to communicate with the ARM processor that is in control of the system. To do so, it will use the communication I2C.

One of things that will need to be relayed to the ARM processor is the final status for navigation. If for some reason the robot can't find a path the user's location, he will need to communicate an error code that will represent "Error in reaching destination". If this is communicated, ARM processor will need to relay this information on to say that the destination couldn't be reached and that the user's items will not be able to be watched. The other option would be that the robot has reached the destination; this will have another code that it will send to represent this. When this is sent, the ARM processor will need to prepare to begin the item watching portion of the algorithm.

Another thing that could be communicated is an error within the system. By this, it is meant that if something goes wrong with the program it will also be relayed to the ARM processor. One such error is if the sensors aren't working properly; if this happens, the navigation system won't be able to work. In this case, the msp430 should send another specified error code. Another possible error that could occur is an overflow of memory from the program being too big and overflowing the allowed memory on the processor. This particular error code will be needed during the testing period. If for some reason the code isn't working, there will be another error code that will be transmitted to let the tester know which element of the system isn't working.

4.4 Wireless System

We have decided to go with a Bluetooth module onboard the microprocessor, linking it to the Bluetooth that comes on every Android phone. Although it is slower and has a shorter range than Wi-Fi, this option is cheaper, and most importantly, easily implementable. Although range is important, most of BroBot's features, including the item watching software, continue to function if the user is outside of Bluetooth range.

The Android device hosting the app will serve as the host of the connection. The microprocessor, using an onboard module, will connect to the host and establish a permanent connection. Nothing will be sent continuously through this connection, but it will be kept open in case the alarm is tripped. In this case, the user will be sent an updated picture of BroBot's field of vision. The user can

choose to cancel the alarm if it has been a false alarm, or can immediately rush back to their things.

```
BluetoothAdapter mBluetoothAdapter=
BluetoothAdapter.getDefaultAdapter();                                    (1)

Intent enableBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);                          (2)

startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);               (3)

Set<BluetoothDevice> pairedDevices =
mBluetoothAdapter.getBondedDevices();                                    (4)

if (pairedDevices.size() > 0) {
for (BluetoothDevice device : pairedDevices) {
mArrayAdapter.add(device.getName() + "\n" + device.getAddress());
}                                                                        (5)

startDiscovery();                                                        (6)
```

These steps show the preliminary commands in Java needed to establish a Bluetooth connection. First, (1) returns the Bluetooth radio embedded in the phone's hardware. (2) tells the Bluetooth that we are going to enable it. (3) enables Bluetooth. (2) and (3) are only called if Bluetooth is not currently enabled on the phone. These commands create a pop-up box that confirm this to the user. (4) creates a set of devices that are already known to the phone, called paired devices. If paired devices exist, (5) iterates through them and adds them to the array adapter. If the device we are looking for, the Bluetooth module, is not paired yet on the user's phone, (6) is called to find nearby discoverable devices. Once the module is found, a connection is established, and we can begin transmitting data.

```
private final BluetoothSocket Socket;
private final InputStream mmInStream;
private final OutputStream mmOutStream;                                  (1)
mmInStream = socket.getInputStream();                                    (2)
mmOutStream = socket.getOutputStream();                                  (3)
bytes = mmInStream.read(buffer);                                         (4)
mHandler.obtainMessage(MESSAGE_READ, bytes, -1,
buffer).sendToTarget();                                                  (5)
mmOutStream.write(bytes);                                                (6)
```

(1) Declares the objects we are using to transmit data. The socket was created earlier when the connection was established. (2) Creates the inputstream object

that we will do our read operations with. (3) Creates the outputstream that we will send out data with. (4) Is the command we use when we read in data from the connected device. Here, bytes is an int representing the number of bytes read in, and buffer is an array of bytes to hold the data received. After reading in the data, (5) is called to signal the app that data has been read in. Finally, to send data out to the item watching software, just call (6).

If the user goes out of range of the connection, the connection breaks and continuously tries to reconnect. If the user powers down BroBot or the app, the connection will be ended.

4.5 Pololu 38 kHz IR proximity Sensor

This IR sensor will be used by the navigation system to ensure that the robot will not run into anything or anyone during its travelling to the user. There will be four different sensors, 1 on each side and 2 on the front of the robot. These modules are quite small at .4” x .6” and need to be situated on the robot so that the modules are parallel with the ground and the sensors are pointing outward.

We are using the 24 inch range IR sensor, though how well the sensors work is dependent on many different things, including the object size, reflectivity and the lighting conditions. The IR sensor uses a 555 timer to drive the IR LED, with this 555 timer comes the ability to change the frequency of the module, which will be set to default for the first implementation.

The Pololu proximity sensor has 4 different connections on the board, these connections are ground, Vdd, out, and enable. Figure 4.5-1 shows the numbers of the sensors that will be on the robot’s body. The logic power lines needs 3.3V to 5V, which is perfect for our system since we will have a 3.3V line. It should be noted that using less than 5V can decrease the brightness of the IR LED, which in turn will decrease the sensing range. If we run into problems we can change our power system to give a 5V line.

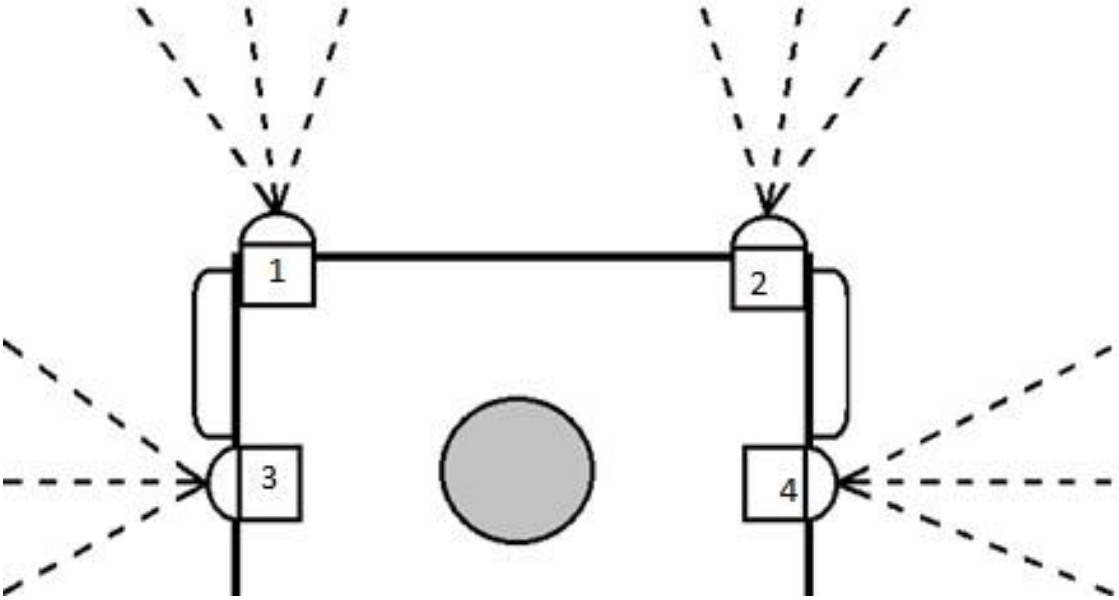


Figure 4.5-1

The out pin normally operates at high, will be pulled low if the TSSP77038 receiver has a sufficient signal. This will be the line that will be needed to go to a GPIO pin on the microcontroller. Since there will be two sensors that will pointing towards the same thing they will share a GPIO pin of the MSP430. Figure 4.5-2 shows the circuit that will be implemented.

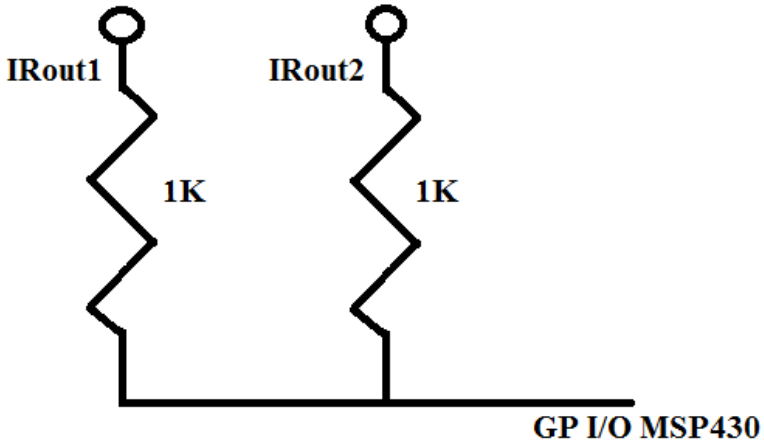


Figure 4.5-2

The potentiometer on the receiver side of the module can be used to change the frequency of the IR emitter LED. This will be tweaked during the testing of the navigation system to ensure that the sensor outputs properly. Changing the frequency can increase the quality of sensing.

Since the side modules will not be used while the front module is used we will need to use the enable pin on the module. This pin is to ensure that other IR sensors will not interfere with each other when using multiple sensors. 2 GPIO pins will used for this operation, one for the front sensor enables, and one for the side sensors enables, though an inverter could be used for this implementation if we run into the problem of using too many pins.

The outputs of sensor 3 and 4 will need to be looked at individually, which means that two more pins will need to be used for this implementation. In total 5 different GPIOs will need to be used in the MSP430 microcontroller. Though we can drop down to 4, if an inverter is used for the enable line.

Since the sensor will not just give a 1 or 0, it will oscillate whenever the sample sense the pulse, we will need to implement a sampling feature in the MSP430. This will be done using an internal clock on the microcontroller, also this will need to be tested during the navigation testing.

4.6 Android Application

4.6.1 Programming Language

The app will be coded in Java. It is the language native to Android, and will provide the smoothest implementation possible. Java comes with many libraries that make GUI frameworks and backend technology alike easier to implement. Libraries such as the Canvas class can be used to create simple buttons and icons to interface with the application. The GUI will be very minimalist.

Java also contains very nice APIs to help interface different components. Detailed in the API section, the Bluetooth API is irreplaceable for our requirement to connect the Android app to the item watching software. Although the members of our team are not experienced in Android development, Java is our most proficient language, so it will be the smoothest learning curve to learn how to implement it on Android.

4.6.2 IDE

The app will be coded in Netbeans, as it gives the greatest support for Java applications. It provides several advantages over Eclipse, allowing Javadocs to be viewed in-console and offering the user much better warning and error detection, along with possible fixes. Netbeans is compatible with the Android SDK libraries that need to be imported to code for Android.

Netbeans' project format is also helpful for the organization of the software. While Eclipse is just a glorified text editor that treats each java file the same, Netbeans

allows code to be sorted into projects. These projects will correspond to the different sections of the code, for example a GUI project, Bluetooth functionality project, and logic project. These projects will interface with each other but work as stand-alone modules as well, providing a robust, modular approach to writing clean code.

4.6.3 Libraries and Tools

We will be using a third party tool called NBAndroid as an attachment to Android. This tool allows for development and testing of Android applications using Netbeans as the IDE. It allows code to be written, debugged, and simulated on an emulator of Android. It also supports directly putting the application on a real Android device.

A second, necessary library that will be included is the Android SDK. The SDK contains all the tools necessary to make a project an Android application instead of simply a Windows application. Once installed, the main libraries will need to be imported in order to properly include necessary functions.

4.6.4 Compatibility

Although this app should run on other platforms, it is being designed for use on more recent versions of Android. The target version will be 4.2 or 4.3 Jelly Bean, depending on which is the current version when coding begins.

Because the app uses external communication with BroBot, it relies on a Bluetooth connection. To use this app, the phone must have this feature, and it must be enabled. If it is not enabled, the app will request the user to enable it.

4.6.5 Communication with hardware

In order to exchange information with the item watching software, the app requires constant (within range) communication with the microprocessor. To achieve this, we are implementing a Bluetooth connection to connect the devices. A Bluetooth connection requires both a host and a device to pair to. The phone running the app will be the host of this connection.

The app will be coded with the Android Bluetooth libraries imported and implemented. This library contains the functions needed to set up Bluetooth for use, find a local device to connect to, and exchange data with it. To do this, the app first needs to create an object representing the phone's Bluetooth adapter. This is done with the function getDefaultAdapter(). Following this, isEnabled() will be called to see if Bluetooth is enabled on the phone. If it is not, the function to ask the user to enable it will be called. After these steps, Bluetooth will be set up on and app and ready to connect.

A set of devices that are known to the app are stored by the phone. This list must then be looped through to see if the device we are trying to connect to, BroBot, is on it. If it is, the connection can be initiated. If not, it will need to be discovered by calling startDiscovery().

4.6.6 Permissions

The Android app must have several permissions to do its job. Both BLUETOOTH and BLUETOOTH_PRIVILEDGED must be set in order to connect to paired devices and pair devices without the user's input respectively. VIBRATE must be enabled in order to allow the app to vibrate the phone if the alarm is triggered. READ_EXTERNAL_STORAGE may be necessary, depending how we decide to handle pictures incoming from the Bluetooth link. If they are stored externally, we will need this to view them and show them on the screen.

4.7 Bluetooth module

The module we have chosen is the RN41 Bluetooth radio. This module is low power and easy to integrate into projects. It uses a UART connection to communicate with the Android app. It runs in low power mode with 250µA used while still being discoverable and connectable. After being connected, it uses 30mA. The module uses a 3.3V power supply, just like many of the other subsystems.

The RN41 has a total of 35 pins. Of these, we will only need a few to connect to the STM ARM processor where this module will be housed. Table 4.7-1 contains a list of pins that we will use on the Bluetooth module.

Pin Number	Use
1	Ground. Will be connected to ground on the microcontroller.
3	Bluetooth master control. We will set this to ground since the Bluetooth module will be the slave.
13	UART_RX. Receive input. Will be connected to the ARM processor's UART output port.
14	UART_TX. Transmit output. Will be connected to the ARM processor's UART input port.
19	GPIO status pin. Reads high when properly connected, low when not. We will read this port to ensure the Bluetooth module is functioning correctly.

20	GPIO auto-discovery pin. Will be set low to disable auto-discovery since the app will connect to it.
21	GPIO status pin. Reads low when connected. We will read this port to ensure the Bluetooth module is functioning correctly.

Table 4.7-1

Pins being read off of the Bluetooth module will be read from GPIO pins on the ARM processor. All pins not talked about are either not connected or not needed for us and will be set to ground.

By default, the Bluetooth module is in slave mode, which we do not need to change. The initial Bluetooth pin code is 1234. It defaults to an 115,200 Kbps baud rate, 8 bits, no parity, with 1 stop bit. Since the module has a USB port, it can be physically connected to the computer to change any of these settings via the terminal. The RN41 can be set to command mode by entering $$$ into the console. This allows commands to be entered to configure the RN41. For our purposes, the default settings of the RN41 are exactly what we are going to use, so no configuration will be necessary.

As long as the module is in slave mode (default), it broadcasts its MAC address and waits for a device to pair to it. The Android app will seek out the address and pair with it. During this time, the Bluetooth master verifies the module's pin code. If it is successful, they exchange a security key. After the pairing is successful and the keys are exchanged, the devices instantiate a connection and they are connected.

The RN41 is a black box. This means we do not need to jump through hoops to use its Bluetooth link. In order to send data from the microcontroller to the app, all we have to do is send the data to the microcontroller's UART output pin. This data is received via the RN41's UART input pin and immediately transmitted via Bluetooth to the app. Conversely, any information received from the app is immediately and automatically transmitted to the ARM processor via the RN41's UART output port to the ARM's UART input port. This makes transmitting data wirelessly as simple as possible.

Figure 4.7-2 gives an example of pseudocode from the item watching software running on the ARM processor. First it checks if the Bluetooth connection is good. If not, it does nothing. It then waits one second so it does not overwhelm any of the data buffers. It sends an updated picture over the Bluetooth link to the app. It then runs the item watching algorithm to determine whether or not to trigger the alarm. If it does, it sends the alarm to the app, which vibrates the

phone and alerts the user to the theft. If the app is out of range of Bluetooth, it has no way of using the connection, but will still set off the audible alarm in a different code segment.

```
while(true){
   if connected{

      delay 1 second;
      UART TX Buffer = currPic;
      if initialpic differs from currPic{
         UART TX Buffer = ALARM;
      }
   }
}
```

Figure 4.7-2

4.8 Power Protection

We would like to give our circuit some protection in the case of a power surge or unintended current. Supply voltage can vary, and PCB components can create a back current. We do not want these currents to destroy our PCB. We can get a fuse to protect our circuit in the event that this happens.

We are implementing a voltage regulator as part of our circuit. This will limit the amount of voltage supplied to the components, but not necessarily the current for reasons described above. If we insert a fuse after the voltage regulator, we can protect the circuit from being destroyed. Since the motors are run by separate AA batteries, they are not a part of the power system and their power requirements can be ignored. The rest of our components are very low-power, with an operating current of less than 100mA. Therefore, we can insert a 100mA fuse between the voltage regulator and the rest of the components.

5.0 Design Summary of Hardware and Software

5.1 Item Watcher Subsystem

5.1.1 Hardware Configuration

While the item watching subsystem does use the microcontroller for all of it computation the ARM microcontroller is really the heart of the project. It will take all of the communication from the user and act accordingly. Because of this a powerful microcontroller with a good amount of functionality was chosen. The ability to take in 8 bit parallel data was a huge reason why the STM32F407VGT6 was picked, also it had a cheap development board.

The camera module that was chosen is very flexible to meet the requirements of the microcontroller. The only thing that isn't easily changed in the camera module is the serial control line which has its own specific protocol, which will make testing more important. But the big thing, the data coming out, has a lot of different transfer options and is easily configured, which is great since the microcontroller only has on specific way to take in JPEG data due to the way it store the data.

The module also has a lot of options when it comes to the compression of the image, which is very important to keeping the size of the pictures small, since initially no external memory will be interfaced. The module can change the quality, the chroma setting, the resolution; basically any feature that one would want to have control over the module lets you have control over. Though these features do come at a price, that price being how difficult it will be to properly interface the camera module with the microcontroller module. The list below shows the overall settings for the camera module, most of these settings can and might be changed from the results of testing this subsystem:

- 4:2:0 Chroma sampling
- 800 x 600 Resolution
- Spoof mode for data interfacing
- JPEG compression
- Quality setting at 20

The spoof mode for data interfacing

To control the camera module there is a two wire serial communication protocol built into the module. This protocol doesn't meet any normal standards built into the ARM microcontroller so the communication will need to be setup using software. The software will use fast switching GPIO pins on the arm microcontroller to ensure a fast enough connection. The microcontroller will act

as the master while the camera module will act as the slave. Since most of the pins on the microcontroller can act as GPIO pins any pins can be used. Because of this a GPIO pin set will be used that isn't being used by any other part of the subsystem.

The camera will be mounted onto an arm coming from the body of BroBot. This is so that the camera will be able to be positioned properly for what space or items it will be watching. The lens will be pointed toward the items that the user wants watched.

Some type of communication has to be made to the navigation subsystem. This communication will include where the subsystem needs to go, and when the subsystem has done its job. Since most microcontrollers on the market have some type of I2C hardware interface built in, I2C communications where decided upon. This decision was to ensure easier interfacing, though the ARM microcontroller has a lot of control and a lot of decisions to make when it comes to the I2C system. The list below shows the overall settings for the I2C communication system between the two subsystems:

- Normal speed mode (100kHz)
- ARM Microcontroller will be the master
- Navigation Microcontroller will act as the slave
- Only one byte of data will be sent during transmissions
- Interrupts will be used to ensure proper communication

For normal operation the ARM microcontroller will be in run mode. In this mode all of the clocks run at full speed and nothing is held back. When the navigation subsystem is on the microcontroller will be in sleep mode. This is so I2C can work but not all of the microcontroller will be on, to conserve power from the battery. The subsystem will wake up the microcontroller and then wait until there is communications via I2C from the ARM processor.

5.2 Navigation

5.2.1 Hardware Configuration

For navigation there are three different types of components that need to interface with the MSP430 microcontroller. The IR sensors will be implemented using GPIO pins from the microcontroller. The output will be sampled using software since it comes in as a continuous digital signal. To communicate with the arm microcontroller I2C serial protocol will be used. This line of communication will be used to tell the MSP430 where the location is, and also will wake up the system after it has been in a low power state.

The last component that is going to be interfaced with the microcontroller are the motor drivers. The motor drivers will be used to have complete control over the rotation of the shaft of the motors. This functionality will be used to easily control how far the robot will go and will be used to keep track of where the robot is.

5.2.2 Steering Mechanics

The chassis that was selected uses four wheel drive. So the four wheels are each controlled independently. This model of steering was chosen because it allows for greater control on ground that could be difficult to travel over. Since the wheels on the chassis are just hard plastic wheels, they wouldn't get as much traction on some surfaces. Seeing that when the robot will be used it may have to travel over different floorings, such as carpet or tile, it should have the ability to still travel just as well on these various surface textures. We also don't want our BroBot to slow down when the surface texture is encountered; we would like for him to maintain a fairly high speed throughout all of his travels.

5.2.3 Algorithms and Interrupts

The algorithm that was decided on for this project uses passive infrared sensor signals as interrupts to detect obstacles, has the ability to turn ninety degrees when it detects an object directly in front of it at too close of a distance, and regularly calculates the distance that has been traveled and how much farther he needs to go until his destination is reached.

Using an algorithm that simply makes turns at ninety degrees angles, rather than slight turns around an object seemed more useful because then BroBot would be completely avoiding the obstacle, rather than slowly trying to move around it. This also allows him to change direction and begin to minimize the distance in the other direction that needs to be traveled. Because the chosen algorithm will regularly track the distances he has traveled and still needs to travel, we could map how much still needs to travel in the two needed cardinal directions. When an obstacle is encountered, he simply needs to decide whether a left or right turn will help minimize the distances to the final point.

5.3 Android Application

Upon start, the user will be faced with a screen allowing them to enter their table number. After receiving the user's location, the app sends a request to BroBot to come. After navigating to the table and being put on it, BroBot will take a picture. This picture will be displayed to the user on the app. If, at any time, the item watching program determines something has been stolen, it will send an updated picture to the user. The user will then have the ability to toggle the alarm off if it was a false alarm.

6.0 Project Prototype Construction and Coding

6.1 Navigation

6.1.1 Sensors

It was decided that four passive infrared sensors with mid-range detection would be used for our project. The next step is to determine where they should be placed on the chassis. Their purpose will be to detect objects which could cause our robot to crash. Our main idea for the movement will be along straight lines, with ninety degree turns when a change in direction is needed. This means that we'll need to be able to detect objects which are directly in front of BroBot, along with things which are to the sides of his movement. The vision of the sides is needed so that he doesn't accidently turn into an object.

To accurately place them, it was needed to know how far away an infrared sensor can detect objects, as well as the lateral range on a sensor. As the typical range for one of these sensors is up to ten meters, but usually closer to five, we will be able to detect from far away. We want to receive a true reading when there is an object one meter in front of him. This will allow ample time to slow down and change direction. The lateral range for an infrared sensor is approximately fifteen degrees in each direction away from the center line. To determine the angle which the sensors should be places at, we need to use simple geometry, given the size of the selected chassis for our project. To detect an object that is one meter away, we can line up the sensors to find the extreme case where the cross sensing areas cross at exactly one meter in front of the chassis. If this were the case, we would need to put two sensors on the front of the robot, on the corners, facing ten degrees out from the centerline. When an object was further than one meter away, both sensors would be on, when it crossed that line though, they would both shut off. This means that nothing would be able to be detected within the range that is less than one meter from directly in front of him. This could work though, because we don't foresee him starting his travels with something directly in front of him.

The next two sensors will need to be able to check for objects located on the sides. The sensors which have already been placed on the front of the chassis will also be detecting on the sides, we need to figure out how far back we can place the last two sensors from the front. We will need to be able to detect things at a closer distance on the sides. We would want to make that we minimize the amount of unreadable area near the front of the chassis. This is because this is where the robot will be turning. While we don't want to line up the sensors exactly, because we would then be sacrificing a lot of distance towards the back, we don't want to place them too close to the back either. If they were placed in the center, they would be able to "see" everything within around fifteen inches of the chassis, and that is the extremes at the corners. If we placed them 0.05

meters back from the front it would be able to make up for some of the vision lost, and it wouldn't hurt the range much for the back either. We mainly need to be able to recognize objects that are towards the front, because we don't want to hit something when it turns.

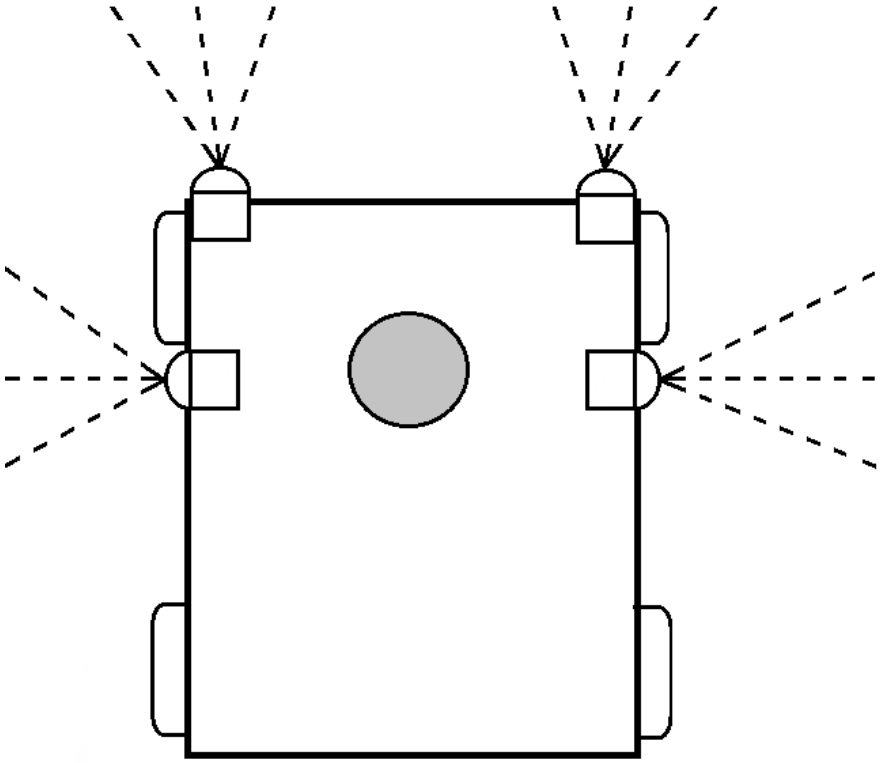


Figure 6.1.1-1 Sensor Organization

6.1.2 Movement

To program the movement of our robot, we will begin by finding all the usable functions and code that is readily available for similar projects. These codes will then be tested with certain values to see how they change as different inputs are given. Different sections of the code will be completed separately; first will be the part of the coded needed simply for using the infrared sensors. We will need to be able to read the signal coming from the sensors a certain number of times each minute. The program should regularly ask for the status of the sensors to determine whether it needs to react to the environment. While composing the part of the code, we would create a variable that would be linked to the sensors' output. This will be used in the code as a function that causes an interrupt if the infrared sensor returns a value higher than some determined threshold; which will mean an object is at a specific distance in front of BroBot.

After this portion of the code has been completed, the motion algorithm will need to be created. The next portion is the actual motion algorithm that was discussed in the research section of the paper. There will need to be two other functions

within our algorithm which need to be coded. The first would keep track of the amount of time which has passed, by using an internal clock. As it keeps track of the time, it will also calculate the distance traveled, based on the average speed of the robot, which we will determine through testing. There will be two variables that keep track of the distance; one that tracks the distance of the straight line, and the other that tracks the overall distance. These will be used so that BroBot can keep track of his location on the map. The final function will use these acquired distances, along with the directions of the turns, to help determine what directions should be traveled until the final goal is reached.

Combining all three of these functions, we should have a complete navigation system. We will have our code running, and just continuing the forward motion if nothing in the system has changed. When an obstacle is visible, as the infrared sensors will signal, then the vehicle should change direction, away from the obstacle. When it changes direction, the new direction should be passed on to the function tracking the location. At this time the direction values will also be updated. So, the direction, distance, and object detection aspects of the movement have been satisfied.

6.2 App Integration

To install the app on a smart phone, Unknown Sources must be checked in Android's settings. Under Settings → Security, check the box called “Unknown sources.” Setting this allows the phone to install apps that were not downloaded from the official app store. Once this setting is selected, simply add the .apk file to the phone and run the Android application manager to install the app.

In order to successfully run the prototype, the app must be installed on the phone. When ran, the Bluetooth module will be activated and begin searching for a possible connection. This connection must be successful for the app to function. Upon connection, it must be verified that data is able to flow from the app to BroBot and from BroBot to the app. To verify the connection, simply summon BroBot. An initial picture of BroBot’s field of vision should appear on the phone. If it does, the connection has succeeded and the app is successfully linked to BroBot.

6.3 Camera

The goal for the camera in this project is to have it be able to view the users items located on some random table. Seeing that the tables in a library vary in heights, raising the camera to the appropriate height is the first task for this aspect of BroBot. Another issue is the orientation of camera, rather than awkwardly placing him somewhere that could cause him to fall and break, he should be able to have his camera adjusted to pointing towards the objects. This

will allow for greater satisfaction for the user to obtain maximum security for their items being watched.

First off, we planned on having the user pick up BroBot from the location he traveled to and placing him on their chair facing the table. This will allow him to be closer to the items, but the chassis is only 105 millimeters high, so it will need to still raise the camera further. To compensate for the lack of height of the chassis, a lightweight tube will be attached to the top of the chassis. Since it needs to be mounted upright, we will use two strong-tie angles, one on each opposite side of the tube. The tube will be located in the center of the chassis. When securing it, screws will be put into the tube, as well as into the appropriate mounting locations on the chassis. The length of the tube will be determined by the length of the camera wires; we will want to leave a few inches so that it won't accidentally pull anything loose.

The next portion of getting the camera correctly lined up with the items is the rotational aspect of the view. The user will need to rotate the camera to direct its line of site to the items. For this we will use a ball and socket joint. On the end of the lightweight tube we will attach a flat plate on a ball and socket joint. To assemble it, there will be the three pieces, the lightweight rod, the connecting piece, and the piece that will hold the camera, which will have the ball and socket joint. There will be a hole in the camera's direct support that will line up with the hole down the tube; this will be used to keep the cords out of the user's sight. Connecting the three components will need to be done before the apparatus is attached to the chassis.

6.3.1 Communication with Camera module

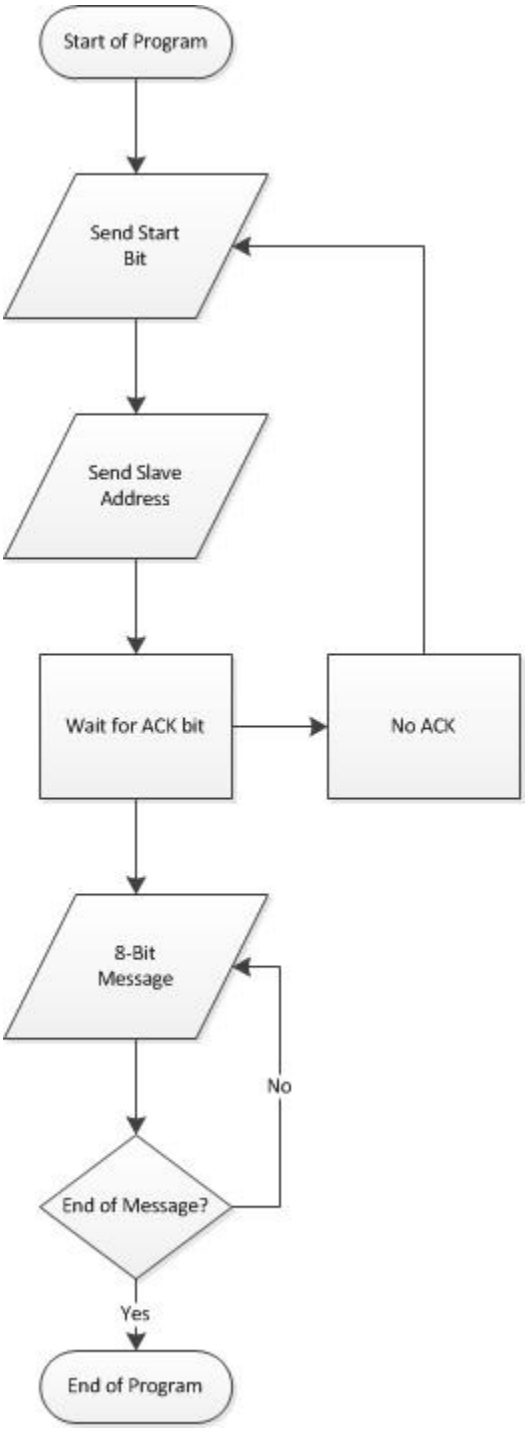


Figure 6.3.1-1 made by Jacob Stewart

Figure 6.3.1-1 shows the flow chart the program will abide by when communicating with the camera module. Since the communication will have to be bit banged a simple program will be written for the communication line. The program will be able to send 8 bit information to write or to read to specific

registers within the microcontroller on the camera module. The program will be written in such a way that registers that will be accessed will have their own variable values; this is to streamline the program so the value won't have to be looked up whenever some type of communication will happen between the two chips.

6.3.2 Data extraction by ARM Processor

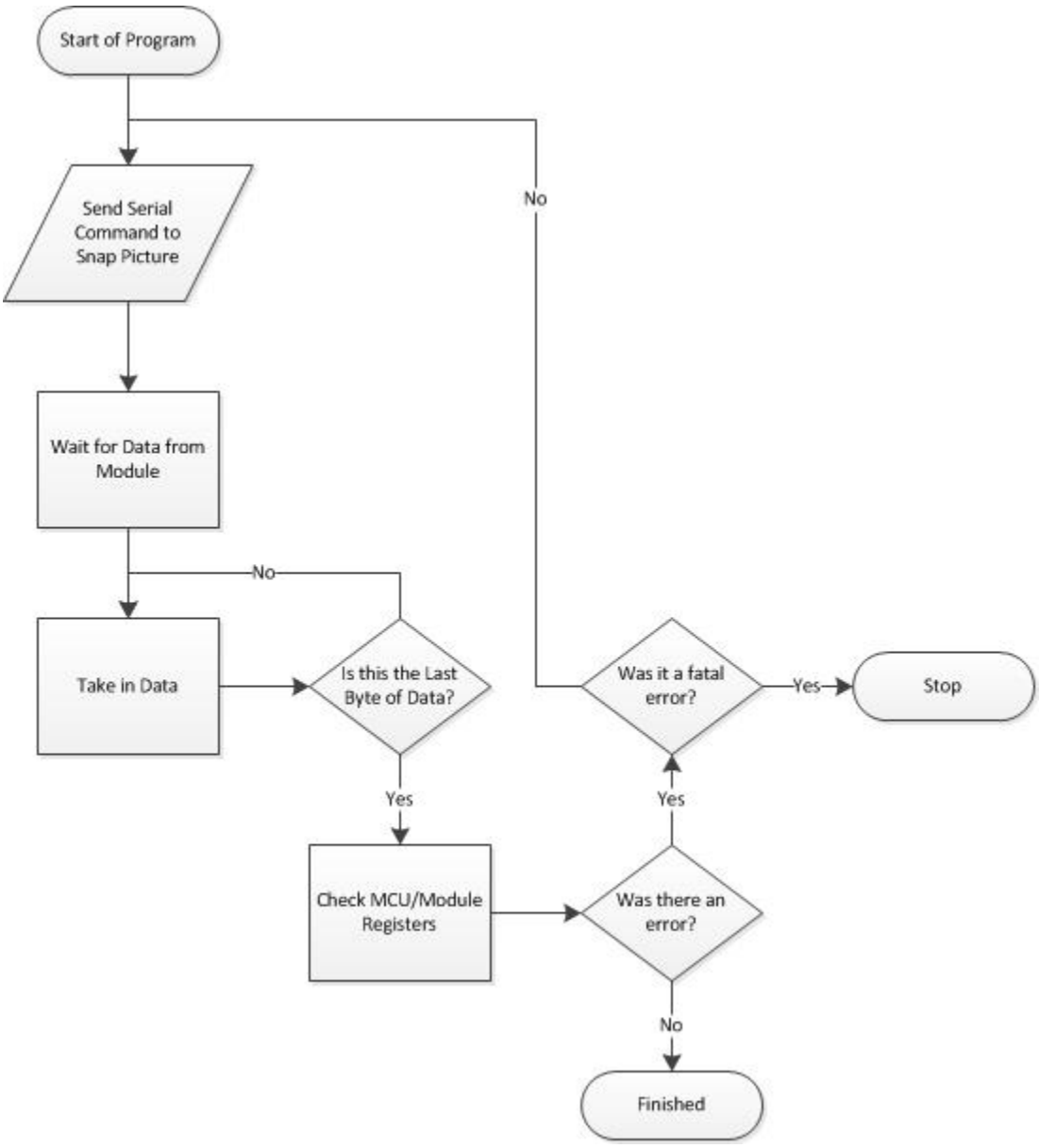


Figure 6.3.2-1 Made by Jacob Stewart

Figure 6.3.2-1 shows the flow chart for the microcontroller when taking in data from the camera module. The chart shows what can happen and when errors are checked, a fatal error can be many things but mostly means that a picture could not be taken in. If the camera module itself never sent anything then the program will stop running, but if there was an overflow due to compression or something else then the program will try to take another picture, after 5 tries of this the program will stop.

Both of these flow charts represent functions within the main ARM microcontroller program. They are going to manipulate data coming in and coming out. They will be implemented along with the item watcher program and will work alongside it to control the data flow.

6.4 PCB

When every system is tested, code written, schematics designed, and everything is working properly, it is time to get our printed circuit boards. This will be used for the final prototype. Our PCB will need to include all the circuitry and processors onto one simple board.

To make the PCB as modular and fail-safe as we can, we will be using a socket to house the MSP430. By putting a socket on the board, we can snap the MSP430 into position, allowing it to be replaced if it breaks or needs to be replaced with another module.

We will be designing our PCB using a program called Eagle. This software allows us to create a visual layout of our PCB based on our schematic and export it to a gerber file that can be sent to a PCB company to physically create. The free edition of Eagle, called Eagle Light, has a few restrictions that we will have to work around.

The maximum size for a PCB is 100x80 mm (4x3.2 inches). Also, only two signal layers can be used, and only one sheet can be created by the schematic editor. Apart from these restrictions, it functions exactly the same as the paid version. Eagle gives permission for anyone using the free version for non-profit, educational purposes to use it, so we will not have any licensing issues.

The core of our PCB revolves around the STM processor used to host the item watching software. Figure 6.4-1 shows the pins used in this processor. These pins will connect to other elements in our PCB and will be the focal point of our design in Eagle. We cannot use a socket for the STM because it is a surface mounted processor.

We plan to order our PCB from www.4pcb.com. Because we are using the free version of Eagle, only two signal layers, the top and bottom layer, can be

designed. Therefore, we have no choice but to create a 2-layer PCB. Normally 4pcb has a minimum of 4 for an order of full-spec 2-layer PCBs, but if you type “Student” in the comments they waive the minimum requirement, allowing us to order only one.

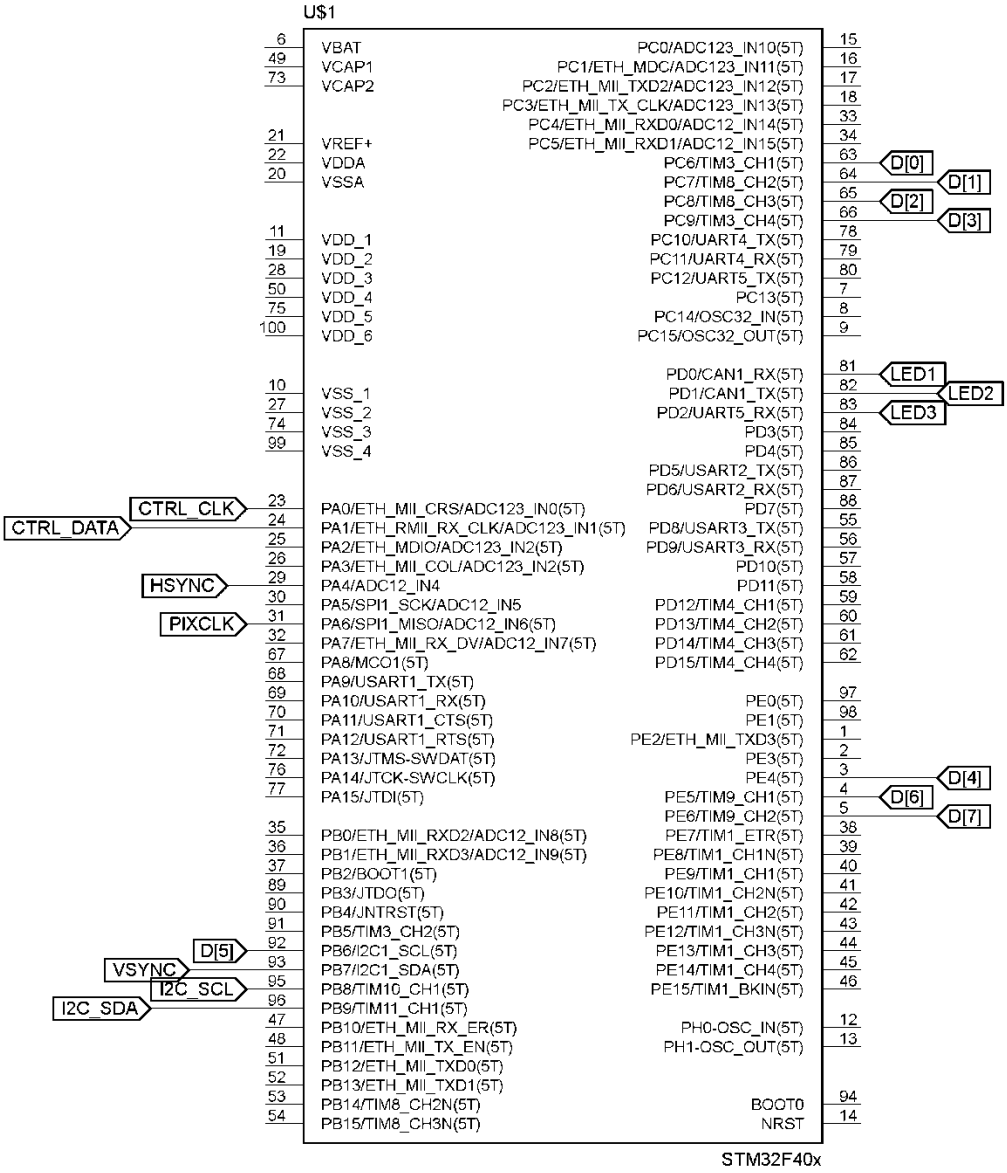


Figure 6.4-1

6.5 Integrating Vision Software

The vision software will be coded on a Windows PC and ported to the processor afterwards. It will be thoroughly tested on the computer using images taken with a webcam. When it is time for prototyping, the microprocessor will be connected via USB cable to the computer. The code will be transferred to a tool called STM Studio, a program designed by the makers of the microcontroller to allow easy

debugging of the software. This software can be used to view all the variables and ensure the program works correctly on the microcontroller.

Further integration will be needed to interface the vision software with the components that need it. First, the Bluetooth connection will need to be configured and the item watching software must be connected to the Android app. This connection must be set up such that the Bluetooth module on the STM processor automatically connects to any users attempting a connection with it, since there is no way to confirm a connection from the processor side. Once this connection is set up, the vision software and the Android app will be able to communicate freely.

On the other end, the vision software needs a connection to the MSP430. We will connect a port from the STM processor to the MSP430 in order to facilitate the transfer of information. We do not need to use too many lines, because the only information that will be passed this direction is the user's location to be used in pathing.

Finally, the item watching software must be interfaced with the camera. The camera will save jpegs taken to the processor's memory. The software must have a pointer to this memory in order to retrieve these pictures. Since we only need to save two pictures at a time, we can hard code the size and starting address of these two memory locations. In this manner, we can point the software to retrieve the pictures from these same two locations every time.

7.0 Project Prototype Testing

7.1 Item Watcher Subsystem Hardware

7.1.1 Camera communication and data flow

Communication between the camera and the microcontroller is essential to the final product working as planned. Therefor a lot of testing time will have to go into this system to ensure that everything is working properly. The first part of this system that needs to be in working order will be the serial communication. Also during this phase of testing a lot of information will be learned about the size of the files coming in, since one of the registers on the camera module will tell you the size of the last picture taken.

The serial communication will control and check different registers in the camera module. The first action that will be done after initializing the module will be telling the camera to take a picture. Once the camera tells the system that this action is finished then the microcontroller will check to see how big the picture is. From there we will change first the resolution from the highest point (1600x1200)

to the small resolution of 200 x 200. Once all that data is tabulated then the quality factors will be checked, this is to see how well the DSP on the camera module can compress the picture image coming into the microcontroller.

All of this picture size data will correspond to JPEG images, which are harder to dissect then raw picture files. If we can find a range of quality around 20-50 where the sizes of the pictures are around 10KB then that would be ideal. But the resolution also needs to be high enough so the picture will not be too hard to grab images from. Once the initial JPEG image data is figured out we will then look at monotone color scheme for jpeg, to see if the difference in color will drastically change the size of the file coming in. Raw image files will also be looked at, this is because they are much easier to perform computations on.

The raw images will go through the same process as the JPEG images, except for the quality factor since there is no compression. All the resolutions will be messed with to see how well the sizes scale with the different settings, the color settings will also be changed to see the different in sizes.

Luckily all of this testing can be done without interfacing the data lines to the camera, which would make the process much more complex. Once all this sizing data is done then we will need to get the data transfer done as quickly as possible. To ensure that the images are correctly transferring we will need to transfer the pictures from the microcontroller to a computer. This can be done through a couple different methods. The microcontroller has a connection to USB through the development board which will be used for this part of the testing.

All the pictures coming in will be looked at to make sure all of the images look how they should.

7.1.2 Communication between subsystems

The I2C system will need to be tested using an oscilloscope and having some way of seeing that the two microcontrollers are talking to each other. This will be done with different LEDs on the two microcontrollers which will show when that microcontroller is sending data and when it has taken the data that it needed. I2C could be tricky to interface correctly since timing is really important in the system, therefore another way to communicate between the two microcontrollers might be needed.

SPI is another interface that works well for this, the only problem is the amount of pins it takes is double the amount I2C needs. This would not be a problem with the ARM microcontroller but might be a problem for the low power microcontroller which will have significantly lower amount of pins. Other than that it would be a good idea to keep this in our back pocket in case we run into problems with using the I2C protocol.

7.1.3 Test LEDs

The test LEDs will be used on the system when BroBot runs into a problem it cannot fix on its own. This would be something like if the picture taken doesn't come in properly into the microcontroller. Or if there is an overflow of data cause by the size of the image, things like this will be shown on the Test LEDs.

7.2 App Stand Alone Testing

Many functions of the app require communication with the microprocessor and input from it, so these functions will be impossible to unit test and will need to be tested with the rest of the components during the prototyping phase.

The main part that can be unit tested is the graphical user interface. When the icon is clicked from the Android main page, the app should open, load properly, and not crash. Displayed should be either a box to enter the table number. When a table number is inputted, a message should be displayed to the user stating that it is unable to establish connection to the item watching software.

When the ARM processor, along with its Bluetooth module, is successfully set up and running, continued testing of the app can begin. Clicking on the app icon on the phone should bring up the app with a field to enter the table number. When this is entered, a button saying "Activate BroBot" should appear. This button is meant to be pressed when BroBot has arrived at its final location and is ready to begin monitoring. When this button is pressed, it should be replaced with an image of BroBot's field of vision, along with a button that says "Stop monitoring." If this is displayed, the test has succeeded. The app should never crash, and if something is not set up properly or ready for use, the user should be presented with a message telling them this.

The deeper link between the app to the MSP430 must also be tested. Although this is not a direct link, information still needs to be able to flow from the app through the item watching software and successfully reach the movement software. This phase of testing cannot be done until near prototyping phase, since we need the Bluetooth link, hard link, and chassis all operating together. Since the MSP430 has no screen or console to display output, we can test this functionality by sending any movement command to BroBot from the app. If BroBot moves, we know the connection is successful. If not, we need to debug the data path and find out where the data is getting lost.

7.3 Image Tracking Testing

The image tracking software can be tested by itself much more easily than the app. While debugging on the computer, an image will be taken by webcam and fed to the software. After this, a second image will be added, very similar to the first image. If the app does nothing, it passes the test. Finally, a different image will be inserted. The app should recognize that this is not the same picture, and activate the alarm. Pictures of increasing similarity can be used to find an appropriate threshold to use as the alarm trigger point. Ideally, changes smaller than an entire object appearing or disappearing should not trigger the alarm, whereas such changes as removing a book or laptop from the scene should. It will take some experimentation to find the exact value to use in order to make this binary decision.

It is harder to test the connection part of the code, and cannot be done until the connections are implemented. The item watching software has two connections – one via Bluetooth with the app, and one via a hard link to the MSP430 and the movement software. To test the Bluetooth connection, simply be in Bluetooth range, open the app, and choose a location for BroBot to go. The app should immediately display a picture of BroBot's vision. If this picture appears, the link works.

To test the connection to the MSP430, both connections from the item watching software must be in place. Again, from the app, select a location for BroBot to go. If he starts moving in any direction, the connection is working. If he does not move, either the connection or the movement software may be bugged, and must be troubleshooted.

7.4 Navigation Testing

7.4.1 Sensors

Infrared sensors don't have guaranteed distance they can detect objects from, so there will be time spent just turning them on and placing objects in front of them, then slowly moving it towards the sensors. We will attach the output pin of the sensor to a multi-meter. Moving the object back and forth from the sensors will change the readings on the meter at some rate. As we will test from the distance directly in front, we will also measure the horizontal range of the view by the same method. Various types of objects will be used for these tests; both dark and light to see how the changes in the color affect how well it detects it.

We will then connect the sensors to the microcontroller and record the different distances it gives for the objects it detects and compare these values with the measured values we find. After we complete the testing for objects we place directly in the line of sight, we will repeat the steps for different angles from the

centerline of the sensors. We will increment the angles by a few degrees each time until we find the boundaries of the horizontal scope of the view. These tested distances will be used to slightly adjust where the sensors are located.

After we determine the information about each of the sensors, we will need to attach them to the chassis and make sure everything still works as designed. We want to ensure that there is no interference between the sensors, as well as with other components of the robot. We will do this by setting up the system and see how the signal is read with no objects in front of any of the sensors. After, we will set up an object in front of one of the sensors and make sure that only the one sensor is picking it up. We will then test it for the rest of the sensors, and follow by placing an object in the range of two sensors at the same time.

7.4.2 Software for Movement

Testing the software for movement will require a few different levels of testing. While going through all the sub-functions of our program, we will test that each piece works alone. The first function will be just one that reads and interprets the signals from the infrared sensors. We will first test the sensors, as mentioned above, and then when we know they're working correctly, we would test them with the code segment. Since we know the sensors are working with the microcontroller, we could verify the code is correctly interacting with the controller, and once verified read the infrared signal and make sure that the code has the correct response to the signals.

The next component of the code is the tracking of time and calculating the distance traveled, based on the speeds of the motors and chassis. We will have to first create some speed to be called the speed of the chassis, and then allow the program to run for some measured amount of time. At the end of this time period it will need to output the distance it calculated. Since we know how to calculate speed from the basic physics' equation, we will know what it should output as the distance. This test will test that the timers are working correctly.

The final piece of the code that should be tested singly is the calculation of the final destination and distance to it. We can track what the output should be by drawing the diagram of the library, or testing environment, and input fake signals that an object has been detected. As we do this we'll have to provide obstacles and distances so that the code will have all the necessary inputs. Once we verify that it can correctly know how far the destination is and the correct turns to make, we can begin the next part of the software testing.

The next thing we'll have to do to test the motion software is a combination of the three pieces of the code. We can do this by just connecting the sensors and simulating obstacles being in the way. At this point BroBot should be able to track the time between interrupts, the distances between interrupts, and the directions that should be turned when an obstacle does appear in the way. We will simulate

all this and make sure everything works as intended, and then only after, we will connect it to BroBot and let him actually make turns and find paths. He should make the correct directional turns at obstacles, as well as do this with ample space before reaching the object, and be able to stop when he reaches his destination.

7.5 Prototype Testing

Although unit tests are important for testing module functionality, when BroBot is complete we will enter the prototype testing phase, at which point we must test the overall functionality of BroBot. Although many of BroBot's subsystems work in a modular fashion, the interconnectedness of each of the subsystems make it very hard to know if the overall system is working properly, regardless of whether or not the unit tests pass.

If the unit tests pass, we can assume the algorithms within each subsystem are working properly. However, we need to test the links and connections between the systems. Powering on BroBot and opening the app allows the test to begin. When the table number is entered, the link between the item watching software and the app will be opened, as well as the link between the item watching software and the app. If the connection fails, the user will be notified by the app. If the test passes, the app will begin to display pictures showing BroBot's vision as it moves.

The unit test for the motion algorithm is very limited in scope as it does not test the data pathways to pass the intended destination from the app to the software. This global test will determine whether or not the data is flowing properly, and also whether or not the motion algorithm succeeds. If BroBot gets stuck on a wall, or does not find his way to an area near the user's location, the motion algorithm has failed. If BroBot does not move at all, the data pathway is not working properly and pathway specific testing must commence.

Once BroBot has arrived near the user, we know that the data pathways are working, as well as the motion software. If the user can see BroBot's view as it travels, we know that the camera interface is working as well. At this point, all that remains to be seen is whether or not the item watching software is working. As this is easily unit-testable, this portion of the test is not that important, but the important part to test is whether or not the item watching software maintains an open Bluetooth link with the app. While in range, the app should receive regular pictures of the items. If the alarm is triggered, whether or not the app is pulled up, the phone should vibrate and give the user warning as well as an updated picture. If these events happen, we know the link is working properly. If they do not, we need to examine the Bluetooth connection for problems.

7.6 Battery Life Testing

To complete our spec, we need to know how long BroBot can continuously operate for. This operating time will vary based on what mode BroBot is operating in. We need to determine how long BroBot can operate in each of his operating modes.

BroBot will use the most battery when he is traveling. All of his systems are active in this mode. The camera is taking pictures, the ARM processor is sending them via Bluetooth, the MSP430 is computing routes, the IR sensors are collecting data, and especially, the motors are consuming incredible amounts of power to move BroBot. Due to these drains, we expect BroBot's battery to drain very quickly in this operating mode. Luckily, BroBot will spend very little time in this mode, since most of the time he will be stationary watching items.

We still want to determine an average running time if he is constantly in motion. We can achieve this by starting a stopwatch at the same time we send BroBot an area to travel to via the app. Before he reaches his destination, we send him a new one, forcing him to keep moving. We repeat this until his battery runs out and he stops functioning. This recorded running time will be considered his average battery life while in motion.

BroBot will consume far less battery when he is in watching mode. In this mode, the camera receives power to take regular pictures, and the STM processor analyzes them and sends pictures and information over the Bluetooth link. However, the motors, easily the most power-hungry component on BroBot, are not operational in this mode. Neither are the IR sensors, or the MSP430 processor. Without these drains, we expect BroBot's battery life in this mode to be several times greater than his battery life while in motion.

To determine his average watching battery life, simply start BroBot with a full battery, and start his item watching functionality from the app. The app will receive regular picture updates of BroBot's field of vision. Leave him running until these pictures stop coming. When this happens, stop the stopwatch. This time is BroBot's average running time when watching.

To determine BroBot's average running time over the course of a day, simply multiply the weighted sums of the two values calculated above using this formula:

$$\begin{aligned} &Average\ running\ time \\ &\quad = \ average\ running\ time\ moving * percent\ of\ time\ moving \\ &\quad + average\ running\ time\ watching * percent\ of\ time\ watching \end{aligned}$$

For example, if BroBot is moving 5% of the time and watching 95% of the time, the equation becomes:

$$Average\ running\ time = average\ running\ time\ moving * 0.05 + average\ running\ time\ watching * 0.95$$

7.7 IR Sensor Testing

An important part of the movement software on the MSP430 are the IR sensors. These sensors are used to avoid obstacles. If the sensors read high, that means they have detected an object in front of BroBot. This can be a person or immovable object. If the IR sensors are not working properly, BroBot can collide into this object, leading to damage to BroBot or worse, personal injury and a potential lawsuit. To mitigate these risks we plan to test the IR sensors to make sure they are working properly before putting BroBot into action.

When a sensor detects an object in the way, its pin reads high. When it does not, the pin reads low. To test the sensors, we must power VCC and set enable to ground. We must then hook up an oscilloscope to the output pin. Because signals get lost, the output will not be 1 constantly while the IR sensor is blocked.

Leaving the sensor uncovered, we will view the output of the oscilloscope. It should read low. The sensor should then be covered. For the duration of the time the sensor is covered, the oscilloscope should read high for at least the majority of the time, with quick dips to low. If this output is observed, the sensor is working.

When reading the output of the sensor into the motion control software, we must sample it at a particular frequency to gather the data. Because the wave is emitted from the sensor at 35KHz, we must sample it at twice that, or 70KHz, according to Nyquist's sampling theorem. Once we have the output, we can check if there are more than some threshold value of high readings per unit time. If this is the case, we treat it as a high reading, and avoid the obstacle accordingly.

8.0 Administrative Content

8.1 Administrative Content Management

Our team consists of four members: Richard and Sarah are Computer Engineers, while Jacob and Anson are Electrical Engineers. The Computer Engineers are focusing more on the software aspects of BroBot, while the Electricals are focused on the embedded aspect, including low-level programming, as well as the power system.

Richard and Sarah together are in charge of the item watching software. As the group members with some computer vision experience, we will take our knowledge of C programming in conjunction with image concepts to complete this segment of the software. Since neither of us have ever designed an image system from scratch, nor do we have much experience writing code from scratch for an embedded environment, we foresee this being one of the harder tasks to accomplish.

Richard alone is in charge of programming the Android app. He has the most experience programming in Java, and is knowledgeable about the tools needed to make a project able to run in an Android environment and use a Bluetooth connection. Although he has never created an Android application before, he is confident in his ability to learn via the resources provided in the Android documentation.

Sarah is in charge of the movement software. She will be interfacing the software with the motors attached to the chassis, and do the location logic involved in the algorithm. Her part of the software will receive the desired location from the app, and will figure out what direction to go in and how to avoid obstacles on the way.

Anson is in charge of the power system. His electrical knowledge will be imperative in supplying each of the subsystems the power they need to operate. He will set up the circuit we're going to use in our final design to print later onto printed circuit boards.

Jacob is in charge of the embedded aspect of the project. He will be writing code to use the Bluetooth module in order to connect the STM processor to the Bluetooth radio on the Android app. He will also be in charge of writing code for the embedded processors to talk to one another, set their clock speeds, and do all of the interfacing required on our two microcontrollers.

8.2 Administrative Content Milestone

The beginning stages of this project has thus far consisted of research. Many interconnected systems are required for BroBot to function, and it is imperative we chose parts in such a way that they interface properly with each other. The movement software on the low-power microcontroller must be compatible with the motors we chose, as well as the high power microcontroller in order to receive direction instructions.

The item watching software must properly be able to connect to the app and the camera. To this end, we had to do extensive research to carefully select our parts and make sure that each was compatible with each other. This also applies to our software. Research had to be done to ensure that our method of communication, Bluetooth, was available for use to connect the item watching software and the app. Research had to be done to find the correct API and make sure it had no restrictions that would make it unavailable for us to use.

Many of BroBot's subsystems can be designed in parallel and implemented at the end of the development phase. However, some components rely on others working properly to be designable. Determining the ones that should be started early requires an analysis of both the time required to design the subsystem and how many components depend on the subsystem functioning properly. Additionally, each member of the group should have a priority item to work on so that the work of other members is not impeded.

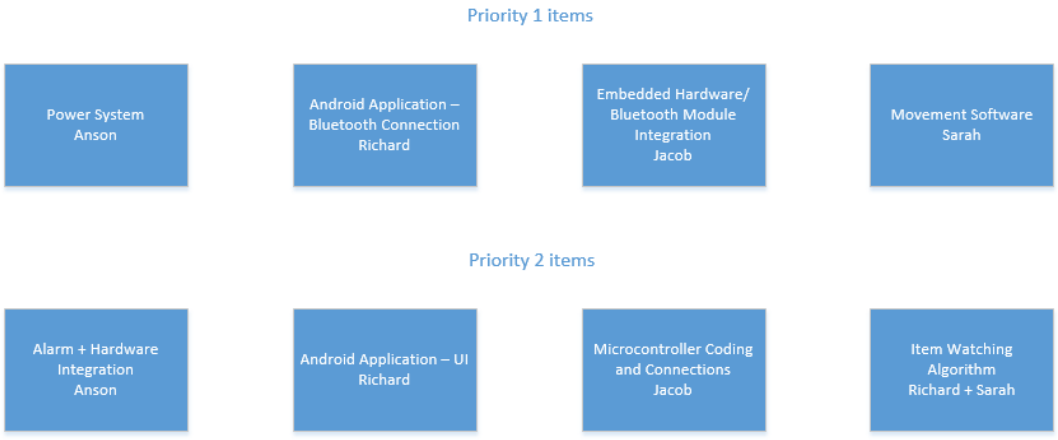


Figure 8.2-1

A few of the subsystems are required to be in place before others can be added, or we foresee will take more time than other subsystems. These we are calling priority 1 subsystems. Subsystems that do not impede the progress and testing of other systems or will be simple to implement are called priority 2 subsystems.

Figure 8.2-1 shows the breakdown of priority subsystems, as well as who is assigned to work on each. Each member of the group should be responsible for one priority 1 system. The first priority 1 subsystem is the power system. Although this can be designed independently of the other systems, it is imperative to have it functional before any other subsystem can be tested. Anson will be in charge of getting this working as early as possible, and it will be the backbone behind the entire project.

The Android application's side of the Bluetooth connection is priority 1 as well. Although the item watching algorithm to determine whether or not two pictures differ is not required to be implemented immediately, the rest of the software, such as opening the Bluetooth link to and exchanging data with the app is priority 1. We foresee having the most issues with the communication between systems, and the sooner we can implement these connections, the better. For this reason, one of Jacob's priorities will be to install the Bluetooth module and write the code to get it to accept incoming connections and correctly transfer data with the app. Additionally, since it is impossible to unit test most of the subsystems, they will not be testable until the communication between the modules is implemented. Also important in the item watching software is the ability to transfer data to the motion control software. Although this connection is not as important between item watching software and motion software, it is still high priority as the motion software cannot be properly tested without being able to receive the desired location from the item watching software.

The Android application is important because of two reasons. First of all, it will take a very long time to code. GUIs are time consuming to set up, and we need to integrate communication along with it to interface with the Bluetooth module. Second, it is the only part of BroBot where the user can control the system. Without a place to enter the user's location, the motion software cannot be tested. Additionally, no feedback from the status of the connection or item watching software can be received. Apart from the audible alarm, which is a low priority item, the app is the only way the user can know if the app is working properly. It also houses the phone's side of the communication software, which is very complex and very important to both testing and implementing BroBot's functionality. Because of this, the sooner the app is in a working form, the better. This will be Richard's high priority item for the beginning of the project. However, the important part of this task is giving the user a place to enter their location and establishing communication with the other subsystems. Therefore, even though the connectivity is priority 1, the user interface itself will be done after as a priority 2 item.

The motion control software will be Sarah's first high priority item. Although no other subsystems depend directly on this software, it will take a very long time to get into a working state. Motion control software is very temperamental and none of us have much experience with it. Although this subsystem receives destination location from the app, it is still important to get a working skeleton completed

early while the communication systems are being designed. This gives us more time to debug and work on what is potentially the most difficult subsystem to implement.

Much of the underlying embedded hardware is necessary to establish communication and functionality across all of the software. Jacob will be in charge of the embedded components. Of these, the communication between hardware is the priority 1 item. Although the app's communication with BroBot is hosted by the app, code must be written for the embedded platform to work with the Bluetooth module on the microprocessor. Additionally, the two microcontrollers, the STM and MSP430, must have a data link between them to share information.

Getting each piece talking to another will be a challenge that must be tackled early in order to test the subsystems, since most of them depend on each other. Although each subsystem of BroBot has its own functionality, it is more important to implement the code allowing these subsystems to communicate with each other before worrying about the subsystem's individual functionality.

Because the app user interface is so paramount to the success of the project, and does not heavily depend on other subsystems, we plan to begin work on that toward the end of winter break. Once the app appears good on the computer, it will be tested on an Android phone to ensure it looks correct. Functionality and connectivity will have to wait until the microcontroller is ready to be connected. Similarly, since the item watching software is the most critical part of BroBot, we will try to begin coding before the semester starts while we do not have other classes in the way.

Since the item watching program can be written and tested independently of the hardware it will be run on, we plan to try to have it working on our computers as early in the semester as possible. Additionally, the movement software is a priority, as it is a topic that we have very little experience with and can potentially take a long time. Once we have the logic set up, the chassis and the motors will need to be connected with the microcontroller to allow this subsystem to be tested as early in the semester as possible.

The item watching software is the most central piece of software in this project. It is important to get communication working between the STM processor to the app and MSP430 as early as possible. However, since the algorithm itself is not very complex, and depends entirely on physical aspects such as how the camera is integrated and where the pictures are stored, it is better to wait until the other subsystems are implemented before writing the majority of this code. For this reason the algorithm itself is priority 2, although the code to communicate with the other systems is priority 1.

We plan to order the parts we need immediately after finals week. Since these parts can take a few weeks to arrive, they will be available to us the moment Spring semester starts, giving us time to do as much as we can before the workload becomes heavy. The priority for the hardware is to get the low-power microcontroller connected to the motor and chassis to test movement software. However, this can wait until the motion software is ready to be tested.

The table below shows our intended milestone chart for this project. The planning phase is based on our effort over the past semester, while the section for the Spring semester is merely a guideline of milestones for us to meet. This table is divided by group member, giving each member of the group a task to focus on. Each subsystem will be designed and coded (where applicable) independently, while keeping in mind interfacing restraints. Multiple unit tests will be performed at this stage to ensure that each stand-alone subsystem works as intended. The final few weeks will be devoted to interfacing all the components together, and extensive system-wide testing.

SEPTEMBER	ANSON	RICHARD	SARAH	JACOB
16-22	RESEARCH: Look for power needs of proposed devices.	RESEARCH: App design; image tech	RESEARCH: Image tech; vehicle control software	RESEARCH: Processor; cameras; hardware
23-30	See above	See above	See above	See above
October	Anson	Richard	Sarah	Jacob
1-6	Continue research	Continue research	Continue research	Continue research
7-13	Update requirements; change focus of research if necessary	Update requirements; change focus of research if necessary	Update requirements; change focus of research if necessary	Update requirements; change focus of research if necessary
14-20	RESEARCH	RESEARCH: Programming the app	RESEARCH: Programming vehicle control software	RESEARCH
21-27	Simulations	Continue research	Continue research	Simulations
28-31	Simulations	Continue research	Continue research	Simulations
November	Anson	Richard	Sarah	Jacob
1-10	Research; simulations	Research	Research	Research; simulations
11-17	Decide on parts	Decide on parts	Decide on parts	Decide on parts
18-26	Finalize paper; order parts	Finalize paper; order parts	Finalize paper; order parts	Finalize paper; order parts
27-30	Study for exams	Study for exams	Study for exams	Study for exams
December	Anson	Richard	Sarah	Jacob
1-8	FINALS	FINALS	FINALS	FINALS
9-15	Wait for parts	Programming	Programming	Wait for parts
16-22	Wait for parts	Programming	Programming	Wait for parts
23-31	Wait for parts	Programming	Programming	Wait for parts
January	Anson	Richard	Sarah	Jacob
1-5	Wait for parts	Programming	Programming	Wait for parts
6-12	Test parts	Programming	Programming	Test parts
13-19	Test parts	Programming	Programming	Test parts
20-26	Assemble	Programming	Programming	Assemble
27-31	Assemble	Programming	Programming	Assemble
February	Anson	Richard	Sarah	Jacob
1-9	Testing	Testing	Testing	Testing
10-16	Debug	Debug	Debug	Debug

17-23	Debug	Debug	Debug	Debug
24-28	Interface	Interface	Interface	Interface
March	Anson	Richard	Sarah	Jacob
1-9	SPRING BREAK	SPRING BREAK	SPRING BREAK	SPRING BREAK
10-16	Interface	Interface	Interface	Interface
17-23	Interface	Interface	Interface	Interface
24-31	Testing	Testing	Testing	Testing
April	Anson	Richard	Sarah	Jacob
1-5	Testing	Testing	Testing	Testing
6-13	Final touches	Final touches	Final touches	Final touches
14-20	Presentation	Presentation	Presentation	Presentation
21-30	FINALS	FINALS	FINALS	FINALS

Figure 8.2-1

8.3 Budget

From the onset of this project, we knew we were not going to look for a sponsor. Since the primary use for BroBot is in a library, we knew companies would have very little use for it, and since it is used indoors, we could not use a solar panel to power it. Since we knew we were on our own, we decided to make budget one of our concerns. Many expensive parts can be replaced for cheaper parts with very little if any decrease in performance.

In addition to the parts that we need for BroBot itself, we need to make use of equipment such as oscilloscopes, multimeters, and a power supply for individual component testing. The senior design lab, accessible by keys given to senior design students, contains these amenities for us to use. Because of this, we do not need to spend money on this equipment.

Additionally, free samples of the STM Discovery microprocessor are available upon request from the company. We plan to utilize this to get the parts we need to complete the working prototype. For our testing purposes, we will need the development board which is not free, so that has been included in the budget.

The chassis we have chosen, a Pirate 4WD mobile platform, it pricier than some. However, it comes with the motors we need for movement, saving us both the expense of purchasing them as well as the potential for incompatibility introduced by trying to put foreign parts together.

Our power system consists of a 4-cell Lithium Ion battery pack, along with the charger recommended by the manufacturer. The battery costs $32.49 at www.all-battery.com, while the charging pack costs $21.95. Although other batteries and chargers can be found for less, this battery is preferred because it has a very high energy density and small size. Additionally, it eliminates one of the major risks of lithium-ion batteries by smart charging. This charger can detect when the battery is fully charged, and will stop the current from reaching the battery. This greatly reduces the risk of explosion inherent in lithium-ion batteries.

The MSP430 gives out free samples of its processor as well. However, we will need a development board to do our testing on. The MSP430x4xx development board costs a ridiculous $175. We will suffice with the MSP430 LaunchPad, which only costs $16.95 on Amazon and $9.99 when ordered directly from Texas Instruments. This board comes with two flashing devices to allow us to load our test software onto the board.

To mount the camera, we need something solid erected on BroBot. We are solving this problem by using a piece of PVC pipe. This requires some supplies such as glue, something to cut it with, a drill, and the ball and socket swivel head. We are using this flexible solution at the end of the pipe to allow the camera to be moved. This gives us the ability to point the camera at the target items regardless of what level they are at compared to BroBot. All these items are easily acquired at Home Depot and we have estimated their price in the budget.

Motor encoders are used to control the rotation of the motors. When given full power, the motor may turn faster than we want it to go and we will not have any control over it. When implementing the motor encoder, we can control to the rotation per minute how fast the motor turns the wheel. This gives us the highest amount of control over our vehicle. The same company that makes our chassis, dfrobot, makes this encoder specifically for our chassis. We will be able to order it online with the chassis and save on shipping.

This budget is just the baseline of our prototype. It does not account for hardware failure of any kind. With a battery as powerful as ours, it is easy to blow out parts and have component failure. This budget can easily grow as the project goes on from these failures and any other kinds of unforeseen difficulties. The budget listed in figure 8.3-1 is only the minimum of the total cost of BroBot.

Item	**Price**
Pirate 4WD Mobile Platform	$49.90
STM Discovery Development Board	$14.90
MSP430 Launchpad	$9.99
RN-41 Bluetooth Module	$21.70
2 Megapixel camera	$18.99
Printed Circuit Boards	$75.00
Report Binding	$15.00
Tenergy Lithium Ion 7.4V Battery	$32.49
Tenergy TLP-2000 Smart Charger	$21.95
IR Sensors	$16.60
PVC supplies	$20.00
Motor Encoders	$20.57
Total	$317.09

Figure 8.3-1

Appendix A

To whom it may concern,

I am a student at UCF pursuing a bachelor's degree in Computer Engineering currently working on my senior design project. While conducting my research, there were a few documents from the website that was relative to my project, such as the LT1121-3.3 datasheet. I would like to use a few of the diagrams within my report for reference and would like to request permission to do so.

Thank you for your time,
Sarah Patten

sarah.patten.35@gmail.com
to micron_inquiries@gmail.com

Permission for Use of images

To whom it may concern,

I am a student at UCF pursuing a bachelor's degree in Computer Engineering currently working on my senior design project. While conducting my research, there were a few documents from the website that was relative to my project, such as the MT9D111 Datasheet. I would like to use a few of the diagrams within my report for reference and would like to request permission to do so.

Thank you for your time,
Sarah Patten

Permission to use images

To whom it may concern,

I am a student at UCF pursuing a bachelor's degree in Computer Engineering currently working on my senior design project. While conducting my research, there were a few documents from the website that was relative to my project, such as the STM32F407VG Datasheet. I would like to use a few of the diagrams within my report for reference and would like to request permission to do so.

Thank you for your time,
Sarah Patten

To whom it may concern,

I am a student at UCF pursuing a bachelor's degree in Electrical Engineering currently working on my senior design project. While conducting my research, there were a few documents from the website that was relative to my project, such as the MSP430x2xx Family User's Guide, and the MSP430G2x53/MSP430G2x13 Mixed Signal Microcontroller Datasheet. I would like to use a few of the diagrams within my report for reference and would like to request permission to do so.

Thank you for your time,
Jacob Stewart

DFRobot Robotics <dfrobotshop@gmail.com>
Mon 11/18/2013 1:30 AM

mark as unread

Hello,

You can, just mark where the pictures come from .

Best,
DFRobot

REPLY REPLY ALL FORWARD ...

sarah patten
Sat 11/16/2013 2:01 PM
Sent Items

mark as unread

To: dfrobotshop@gmail.com;

To whom it may concern,

I am a student at the University of Central Florida pursuing a degree in computer engineering. I am currently in my senior design course, working on a research and design paper, and would like to request permission to use a few images of products from your website. The images in particular are of the Baron-4WD Mobile Platform, Pirate-4WD Mobile Platform, and the Cherokey 4WD Mobile Platform.

We are currently researching the chassis model we would like to use in our final project, and both of these have been under consideration.

Regards,
Sarah Patten

sarah patten, Nov 17 02:56 (HKT):

To whom it may concern,

I am a student at the University of Central Florida pursuing a degree in computer engineering. I am currently in my senior design course, working on a research and design paper, and would like to request permission to use a few images of products from your website. The images in particular are of the 4WD Robot Chassis (KIT) and the Aluminum 4WD Robot Chassis - Gold(KIT).

We are currently researching the chassis model we would like to use in our final project, and both of these have been under consideration.

Regards,
Sarah Patten

Rich, Nov 17 20:09 (HKT):

Dear Sarah,
Thanks for contacting the HobbyKing Support Team.

I do not have the authority to make that decision but once we are credited I do not foresee a problem.
I will forward your request and unless you hear otherwise consider it sanctioned.

Rich

Thanks for emailing support.
If you have any other questions, please let me know.
Best Regards.
Rich
HobbyKing Product Specialist
Dont forget to follow us on Twitter and Facebook for discounts and promotions!
http://www.facebook.com/HobbyKing
www.twitter.com/hobbykinglive

Appendix B

Micron Technology technical staff, *MT9D111 Datasheet*, Micron Technology, 2004.

Pololu Robotics and Electronics technical staff, *Pololu 38kHz IR Proximity Sensor, Fixed Gain, High Brightness*, Pololu Robotics and Electronics, 2013.

Roving Networks technical staff, *RN41/RN41N Class 1 Bluetooth Module*, Roving Networks, 2013.

STMicroelectronics technical staff, *STM32F407VG Datasheet*, STMicroelectronics, 2013.

Texas Instruments technical staff, *MSP430x2xx Family User's Guide*, Texas Instruments, 2013.

Texas Instruments technical staff, *MSP430G2x53 MSP430G2x13 Mixed Signal Microcontroller*, Texas Instruments, 2013.

Appendix C

Eagle

Microsoft Visio

Microsoft Paint