UNIVERSITY OF DETROIT MERCY

Microcontrollers Final Report

ELEE 3860/5086, 3870/5087

Faculty: Dr. Paulik

By Wento Bi, Ziyu Wang, and John Slowik

12/6/2016

Abstract

This report recounts important aspects of our development of an autonomous robot with colored object tracking, collision avoidance, and data communication functionalities. Overall, this development focused on the development of module interfaces rather than modules themselves.

Contents

Introduction 3
Objectives 3
Initial Plan 3
Context Analysis 4
Initial Design Constraints 4
Major Revisions 5
Major Technical Problems 6
Parts Failures 6
System Outline 7
Capabilities 8
Hazard Detection Fidelity 8
Object Tracking Refresh Rate 8
Object Avoidance Efficiency 8
Limitations 8
Object Avoidance Routine 8
Battery Life 9
Low Level Object Avoidance 9
System Details and Design Rationale 9
LIDAR Interface 9
Pixy Camera 10
Arbotix M 1 & Servo Interface 10
68HS12 – Main Control Algorithm 10
XBee Communication 11
MATLAB Display 11
Arbotix M 2 & Hexapod Chassis 12

Power Supply 13
Security Case 14
Software Architecture 15
Timing Analysis 16
Control Algorithm 16
Possible Improvements 17
Summary 18
Workload Distribution 18
Conclusions 19
References 20

Introduction

Vaguely put, the goals of this project were to "explore C and assembly language control structures and to develop an understanding of the features associated with the 68HS12 Interrupts, Serial (SCI), and other interface modules which could include PWM, SPI, I2C, and Timer modules". In general, the project revolved around solving and implementing interfaces between modules, with hardware design and robot behaviors being the most open-ended parts of the project.

Objectives

The following requirements were presented as semi-vague objectives for the project:

- Modify a Trossen Robotics Chassis to function as the project's frame and drive train.
- Utilize a LIDAR-Lite V3 and a Dynamixel to continuously scan the environment.
- Transmit this environmental data to a base station computer where it can be displayed.
- Using the data, implement a wide variety of behaviors, including tracking, following, avoidance, and obstacle avoidance.
- Graduate students will also implement a Pixy Camera to further assist unique behaviors.

Initial Plan

We decided early on that we should follow the recommended project completion milestones; in spite of a significant set-back early on, this gave us a solid idea of our overall progress throughout the report. The milestones we followed were the following:

Milestone 1: Hardware design and implementation; Chassis Interface

Milestone 2: Power Systems; Servo Interface; Start Pixy Camera Work

Milestone 3: LIDAR I^2C or PWM Interface; Complete Pixy Camera Interface

Milestone 4: Integrate Systems together; Write Behavior Routine

Final Milestone: Write Report/Poster; Complete Robot

Unfortunately, we found that our project plan proved insufficiently robust, which was stressful as the project came to a close. This is discussed more under Major Revisions.

Context Analysis

To start the project off, we were presented the project goals and objectives as stated above. We were also presented a basic Milestone outline, and an outline of some of the systems that we would be using. The components we were presented were as follows:

- **Trossen Robotics Hexapod Frame:** This is our robots frame; it came equipped with an Arbotix M Board, and had a well-developed controller-based walking program built into it.
- **Pixy Camera CMUcam 5:** This is our robots primary sensor; default software for a desktop computer was available, as well as a relatively complete C interface code for interacting with the camera. The primary challenge presented to the graduate student was in understanding this interface and having it work smoothly with the rest of the program.
- **LIDAR Lite V3:** This is our robots secondary sensor; it is a Light Imaging, Detection, And Ranging system. It can provide the distance to an object using either an I^2C or PWM.
- **Dynamixel MX-28 Servo**: This is a very precise servo designed to rotate the LIDAR and provide 360 degrees of vision. It is compatible with the Arbotix M driving similar motors in our robots Drive Train.
- **68HS12 FreeScale Techarts Board:** This is the core of our robot; this microcontroller board was to control all major aspects of the project. We had been using a larger variant designed for debugging until this point to learn assembly and C coding. A large amount of documentation was provided for this Microprocessor.
- **XBee Transceivers:** We were also provided two XBee Transceivers; these allow our robot to communicate with the base station wirelessly.
- **2 Arbotix M Boards:** Two Arbotix M Boards were provided for our design; one came with the Hexapod Chassis. The other Was made available to more easily interface with our Dynamixel Servo.
- **3D Printing Capacity:** It was made apparent that we would be able to design and print 3D parts in our Lab to customize our Chassis to do exactly what we needed it to.

Overall, our team was well prepared to program the 68HS12, but we were less prepared to to interface with the many part of this project. We had also yet to cover basic communication routines in either our lecture or lab.

Initial Design Constraints

The hardware provided above, while in general not binding, were roughly implied as being constraining factors on our design. Another important constraint was our battery; although our battery is a powerful 11-12 volt power supply, it also presented both a spatial and temporal limitation, as a large weight for our robot to carry around and as a supply that could run out.

We also found that our Pixy Cam was an interesting problem; we found that it's armature combined with our mounting routine made it very tall. This meant that we needed to mount it in such a way that it wouldn't conflict with our LIDAR.

The amount of flat space on our robot was an important constraint, considering the amount of components we needed to mount to it; adding extra layers and considering how we should allocate that space on our robot was one of our first design considerations.

Major Revisions

Overall, there were no major revisions to our original project plan. One revision worth noting is that we had initially anticipated using I^2C to interface with our LIDAR. After we arrived at that point in the project, we found that we were pressed for time and the PWM interface was sufficient for our needs.

In hindsight, however, several changes would have benefitted our project greatly. To begin with, the final stages of the project would have been much better served by a greater focus on systems integration, documentation, and coordination. With respect to system integration, we probably allowed ourselves too little slack near the end of the project assuming all of our systems would just magically work together. Ensuring that the modules we developed each milestone actually worked in tandem would have been a substantially safer and more balanced approach.

Documentation, on the other hand, would have better served some of our debugging as well as in writing the report; getting pictures of our work for our poster was something that took a surprising amount of time that would have been better distributed over other development hours. It is also important to save code iterations and configurations that work, so the project always has something to fall back on.

Finally, our team experienced traumatizing coordination issues. One example is that different teammates sometimes developed the same subsystem simultaneously; our power system, for example, had been entirely designed but untested by one member before another decided that the apparent lack of progress meant that they should simply implement and assemble another design. This cropped up again with code development for the LIDAR system. Another example is that a certain team member went on hiatus to another country for 10 days in roughly the middle of the project. Although they were reachable, the team plan did not sufficiently accommodate the sudden absence, and only a single member remained productive for the duration.

Unfortunately, although both of these problems could have been resolved with better communication and coordination, the lack thereof exacerbated their consequences, meaning the project was harder than it needed to be. Interestingly, although these problems were overcome by individual investment in the project, they can also be attributed to a lack of individual investment. Overall, these problems primarily resulted in very tight failure margins leading up to milestones and ultimately the deadline of the project.

Major Technical Problems

We had anticipated that the project would present technical difficulties. Unfortunately, it is unclear what the cause of many of them was. Overall, poor wiring may have been the primary result, but unfortunately all of the failures were analyzed after their point of failure, and as such it can only be determined through speculation.

Parts Failures

Arbotix

All of the major failures in our project were part failures. To begin with, after the successful development of our chassis interface we were performing some final system demonstrations when our chassis completely relaxed and became unresponsive. Since it was very late and the primary coding problem had been solved, the system was turned off and debugging was left for the following day. That morning, the system was completely deconstructed and it was found that the chassis' Arbotix board had heated up a very great deal, and was assumed to be the problem. Unfortunately, in order to determine this, the entire system was deconstructed, so either faulty wiring or a faulty board could be to blame.

LIDAR Failure

Initially, the main architect of this module spent many, many hours attempting to debug his code and validate its functionality. The problem escalated to the point where the whole group become focused on this one task; at this point, hardware debugging started. First, the trigger output was proved functional using an oscilloscope. The oscilloscope didn't, however, register any LIDAR outputs, which we found strange. It was then surmised that the cables could be the problem, as the LIDAR output cables had been shrunk wrapped to our wire slip; using an ohmmeter, this was also found to not be the problem. Then, after having another group set up their LIDAR in a passive operating state, we connected our to the functioning setup and found it unresponsive. Although the failure cannot be determined for certain in this context, bared and unsecured power wires could easily be to blame for this failure. On the other hand, because of how this piece of equipment needed to be set up, strictly incorrect wiring could also have been to blame. And again, barring both of these possibilities, the piece of equipment itself was not tested in the initial module stages, and itself could have been to blame.

SCI1 Port Failure

This problem was found after some investigation into our servo data routine; after the routine was run on another microprocessor board and determined to function perfectly, the SCI1 port was declared at fault. Again, this resulted after extensive use of the equipment piece in question, and the pin was not initially tested for functionality, so the problem could again lie anywhere between bad wiring and a bad board to begin with.

System Outline

The system architecture we used was essentially pre-ordained by the project's provided hardware. Essentially, the design consisted of a communication routine for each individual module in the 68HS12; in order to finalize our robot, we need only amalgamate them.

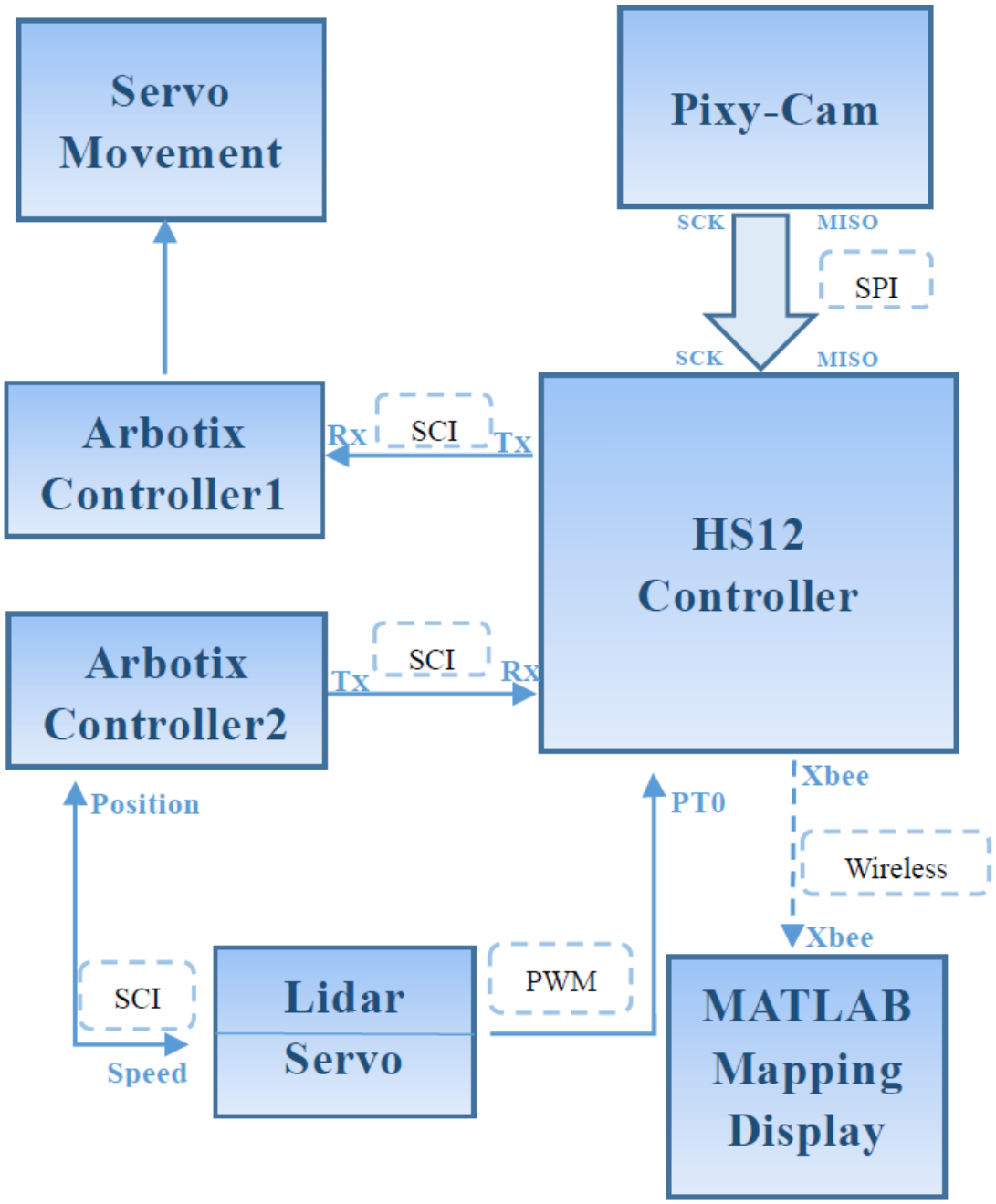


Figure A: Module Map

Capabilities

The system we've designed meets the core requirements of the project: it is capable of searching an area for a color, track and follow that color in a smooth manner, perform basic object avoidance for the duration, and transmit basic environment data to a base station where the data is displayed.

Hazard Detection Fidelity

By making use of its LIDAR, our robot is capable of avoiding most tall objects. The resolution of this system at its default rate, we found, is approximately 2 centimeters wide twenty centimeters away from the robot. This means that our detection apparatus should be able to consistently detect the presence of such objects at the furthest warning distance of our navigation routine. It is reassuring to know that, if we had the time, the sensor we are using would allow us to substantially improve our navigation routine, possibly to the point of mapping entire areas with decent fidelity.

Object Tracking Refresh Rate

Under normal conditions, our robot's camera doesn't update quite fast enough, resulting in choppy tracking behavior. However, as the result of a specific configuration on our system, our Pixy camera is capable of refreshing at a much greater frequency. This is a tangible benefit to our robot, as it allows the system to track objects in its vision range more effectively.

Object Avoidance Efficiency

In order to effectively navigate its environment, or robot takes in between 300 and 600 point of data a second. This feat in and of itself is not significant, but in order to perform object avoidance our robot must also be persistently aware of about two thirds of these points, and must respond accordingly. Although the weight of this computation is not terribly great for our processor, we found that we could reduce it by splitting the data into one of several categories and simplifying the result. Because this algorithm allows us to immediately scrub the data for its usefulness in our algorithm, we can promptly delete it, and ultimately complete the same task in a fraction of the computation time. Overall, the time saved from processing the data in this manner might save us roughly 4/5's of the computation time! This extra time can be spent elsewhere, improving our systems sensor fidelity and object tracking response speed.

Limitations

Object Avoidance Routine

To begin with, our system's object avoidance routine and search routine have a rudimentary design at best. Regarding the avoidance routine, the system is programmed to only strafe into freely navigable adjacent to the target. Thus, although it may be able to find its way around a water bottle, it is not likely to be able to navigate out of a trap; this limitation could be somewhat resolved by implementing distance measurement regarding possible targets. Regarding our system's search routine, the system is not programmed to search a space in an organized manner; instead, it will choose to rotate a semi-random amount, pan its camera around, walk a random amount, and pan its camera again. This loop repeats as necessary until an object is found; although the robot will not walk into

objects during this routine, the routine does not take into consideration environmental hazards when determining a new direction to check or walk to.

Battery Life

Another limitation is our system's battery life and accessibility: it is limited to between ten and twenty minutes when it is in full operation (i.e. scanning, tracking, and walking around). Charging the battery requires either removing the battery entirely, which entails inverting the robot and removing four screws, or placing the robot almost directly atop the charging apparatus due to the short length of an associated battery data cable. Although there may be a redesign in the immediate future to circumvent these problems, they are present in the current design scheme.

Low Level Object Avoidance

Another limitation is low level object detection; due to the elevation of the robot's LIDAR, the system cannot detect collision hazards lower than about 10.5 inches. Although this is beyond the basic requirements, it is worth noting that as a result of this our system will not avoid low objects and is limited to only avoiding tall ones (which coincide with its plane of vision). Again, because of the detection rate, these must be at least two centimeters wide.

System Details and Design Rationale

LIDAR Interface

For our LIDAR interface, we decided that the I^2C communication routine was going to be too complicated, so instead we ended up using the PWM interface. The PWM interface sends out a 10us/cm pulse; as part of our routine, we catch the leading edge of this pulse and measure it until the falling edge. Our design will cut off the LIDAR at 5meters, since we would like to guarantee a 100Hz data capture, but it does not currently. If we do not cut off the data send, then the LIDAR will take too long to send the next distance. Operating the PWM communication channel requires using the 68HS12's timing port T; port T has the integrated features to detect and measure incoming pulses.

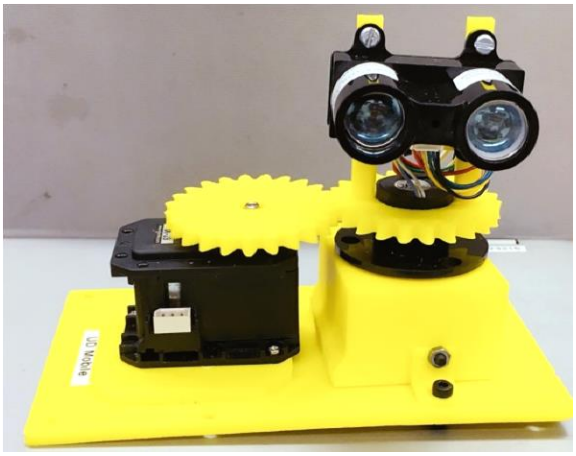
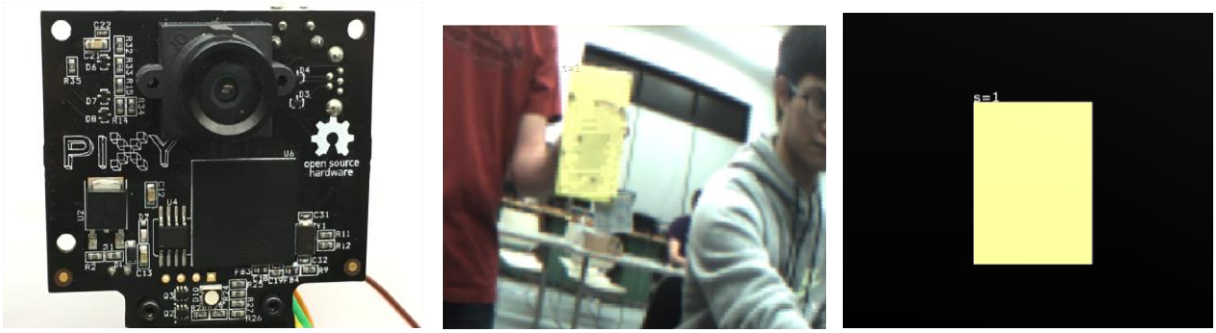


Figure B: LIDAR Mounted to Servo

Pixy Camera

For the pixy camera, we used an SPI interface that initially tells the camera to report the x position of the largest block, as well as to set the servo positions to that blocks current position. The in order to operate the SPI interface, we have adapted code from last year that does a majority of the work for us. Overall, we send data on the Master Out line while maintaining a clocked line; once this data is received, the Pixy Cam will respond on its own line. This is probably half Duplex communication.



Figures C, D, and E (Left to Right): Pixy Cam, Target Object, Detected Color Block

Arbotix M 1 & Servo Interface

For the first Arbotix Servo interface, we use the input of SCI1 to receive the data on the 68HS12. Notably, we simply have the Arbotix continually send data to the 68HS12. As a result, the 68HS12 can decide to retrieve the data at any time, and no return communication is necessary.

68HS12 – Main Control Algorithm

Our main system must first perform initialization routines when it boots up; this includes setting up ports and initializing SCI and SPI communications. When this completes, it enters a main loop – it starts by gathering data, first from the Pixy Cam, then from the LIDAR (this requires some waiting), then from the servo (via Arbotix M). As soon as its gathered the LIDAR and Servo Data, it sends it to the Base Station, and then processes it into one of the possible movement regions. If its close enough, the region

it's placed into is incremented and has its flag set to UNSAFE; once the increment reaches a between 4 and 8, depending on how many data points are in the region, the flag and value will both reset. Finally, the Robot will determine

XBee Communication

We used two Xbees to communicate with our MATLAB interface. First we needed to use XCTU to configure the two Xbees to ensure they worked. Then, we used the serial port on our HCS12 to send data. We initialized the serial port (SCI0), to which one of the Xbees was attached. This Xbee would then send our package the other Xbee. Finally, we can read the data in our computer's COM that the Xbee is connected to, giving MATLAB access to it.

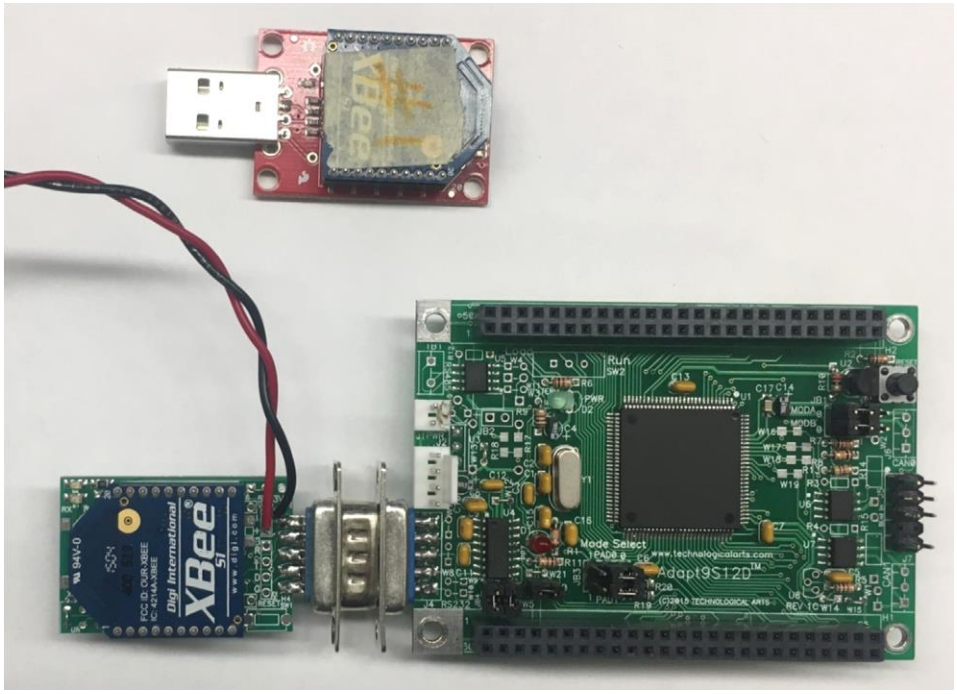
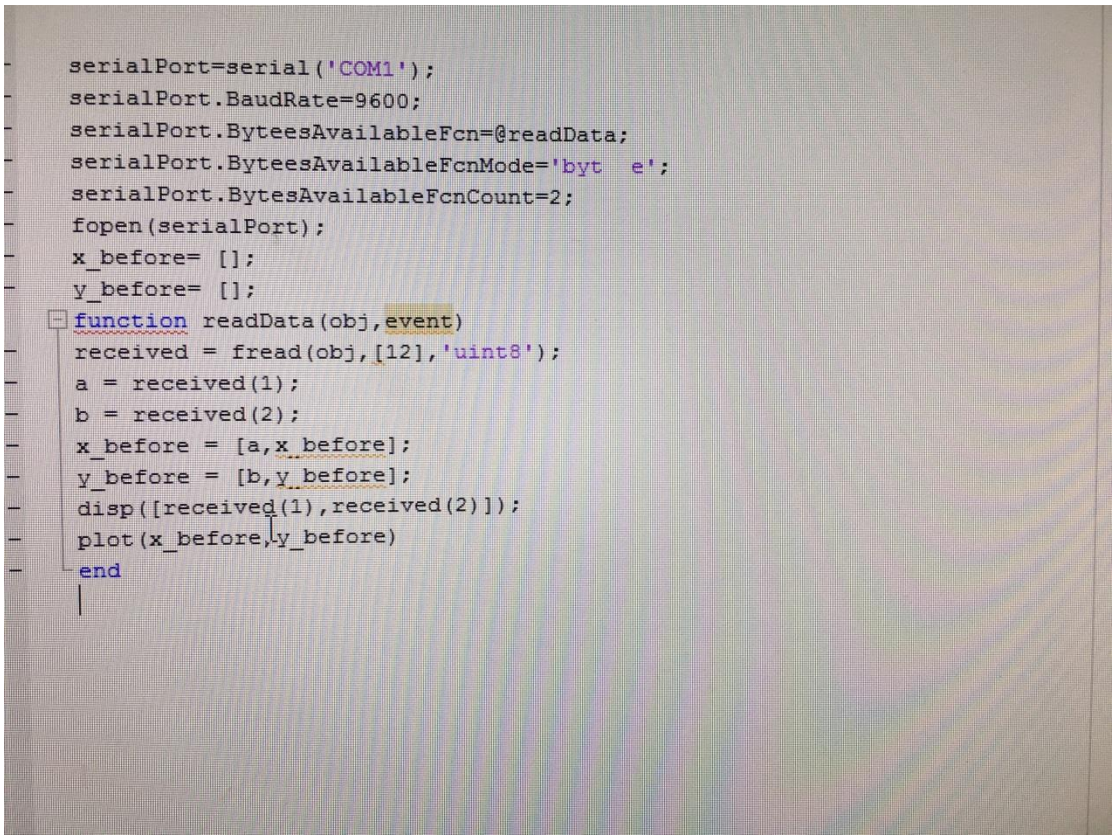


Figure F: Xbee's Attached to SCI1 and USB Pad

MATLAB Display

We send our LIDAR distance and the servo angel(cosine) through the Xbees and then we construct an image out of them in MATLAB. In MATLAB we open the serial port to read the data and we design a matrix to store the data otherwise the plot function can on it plot one point every time. Then we plot the matrix in MATLAB interface. By this we can get the polar coordinates. In our final design, we will configure the matrix to restart after 100 data points are received.



```
serialPort=serial('COM1');
serialPort.BaudRate=9600;
serialPort.ByteesAvailableFcn=@readData;
serialPort.ByteesAvailableFcnMode='byt  e';
serialPort.BytesAvailableFcnCount=2;
fopen(serialPort);
x_before= [];
y_before= [];
function readData(obj,event)
received = fread(obj,[12],'uint8');
a = received(1);
b = received(2);
x_before = [a,x_before];
y_before = [b,y_before];
disp([received(1),received(2)]);
plot(x_before,y_before)
end
```

Figure G: MATLAB Code Screenshot

Arbotix M 2 & Hexapod Chassis

Power Supply

Due to an absence of communication, our group designed and implemented two power supplies. More can be read on this under Major Revisions.

Implemented Supply

This supply was designed with some assistance from Dr Paulik; it has several notable features. Most importantly, it has a voltage switching power regulator that outputs 8 and 5 volts from an 11 volt supply.

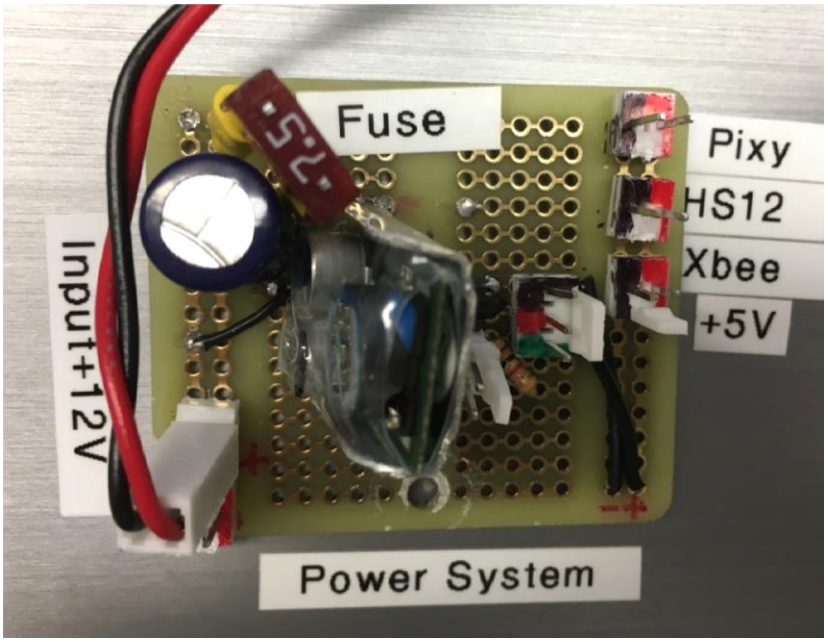


Figure H: Implemented Power Supply Image

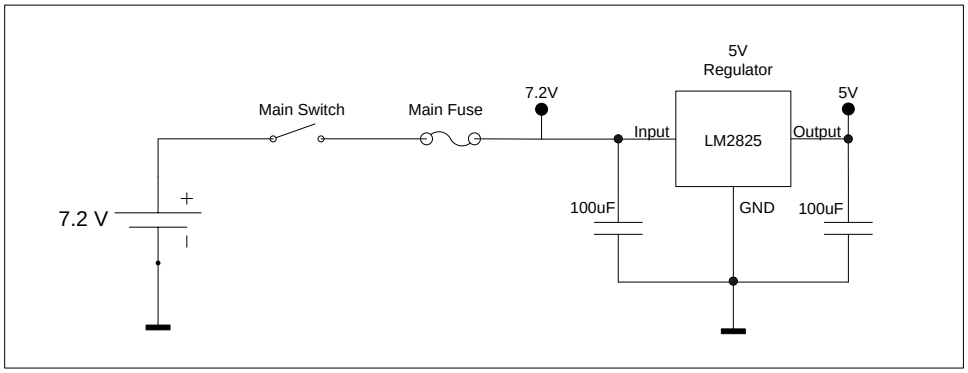


Figure I: Implemented Power Supply Schematic

Note: The diagram above should contain a 1000uF capacitor on the left, and a3000uF capacitor on the right. In our implementation, we used 780uF and 2800uF, but the important thing is that the right capacitor be at least 3 times as large as the left.

Unimplemented Supply

The standard models are equipped with 7.2V power supply, and MCU system, path identification of photoelectric sensors, optical encoder, etc. all need 5V power supplies. The servo motor operating voltage ranges from 4V to 6V; in order to improve the servo motor response speed, we used a 7.2V power supply. We have a series power supply, linear regulator power supply (LM2940, 7805, etc.), and switching power supply (LM2596) two categories. The LM2940-5 supplies power separately for the single-chip. Because other modules need to pass a large current, the use of LM2940-5 and LM2596-5 on the control system and the implementation of partial power supply can effectively prevent the occurrence of the device interference and the current shortage of the problem, making the system to work steadily.

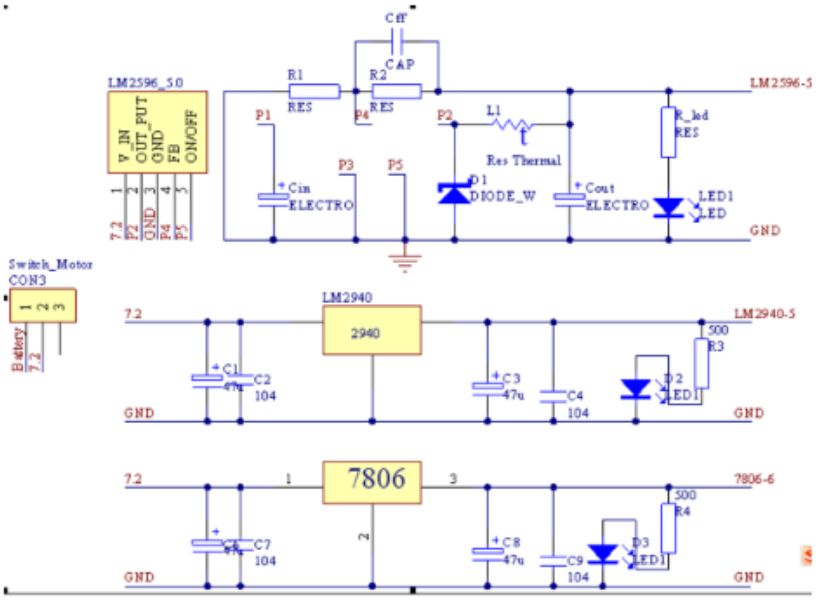


Figure J: Unimplemented Power Supply Schematic

Note: Because this is a simulation, these exact components may not actually be available in practice.

Security Case

Although not included in our project plan, it was later made apparent that a secure case was needed to surround the critical systems of the robot. When our robot is in its final stages of integration, we will design and implement firm panels to secure it as necessary.

Software Architecture

Our software architecture followed a very simple methodology and routine that imitates the modular nature of the system. In general, individual modules were developed individually, then pieced together. Ultimately, this resulted in the full system arrangement.

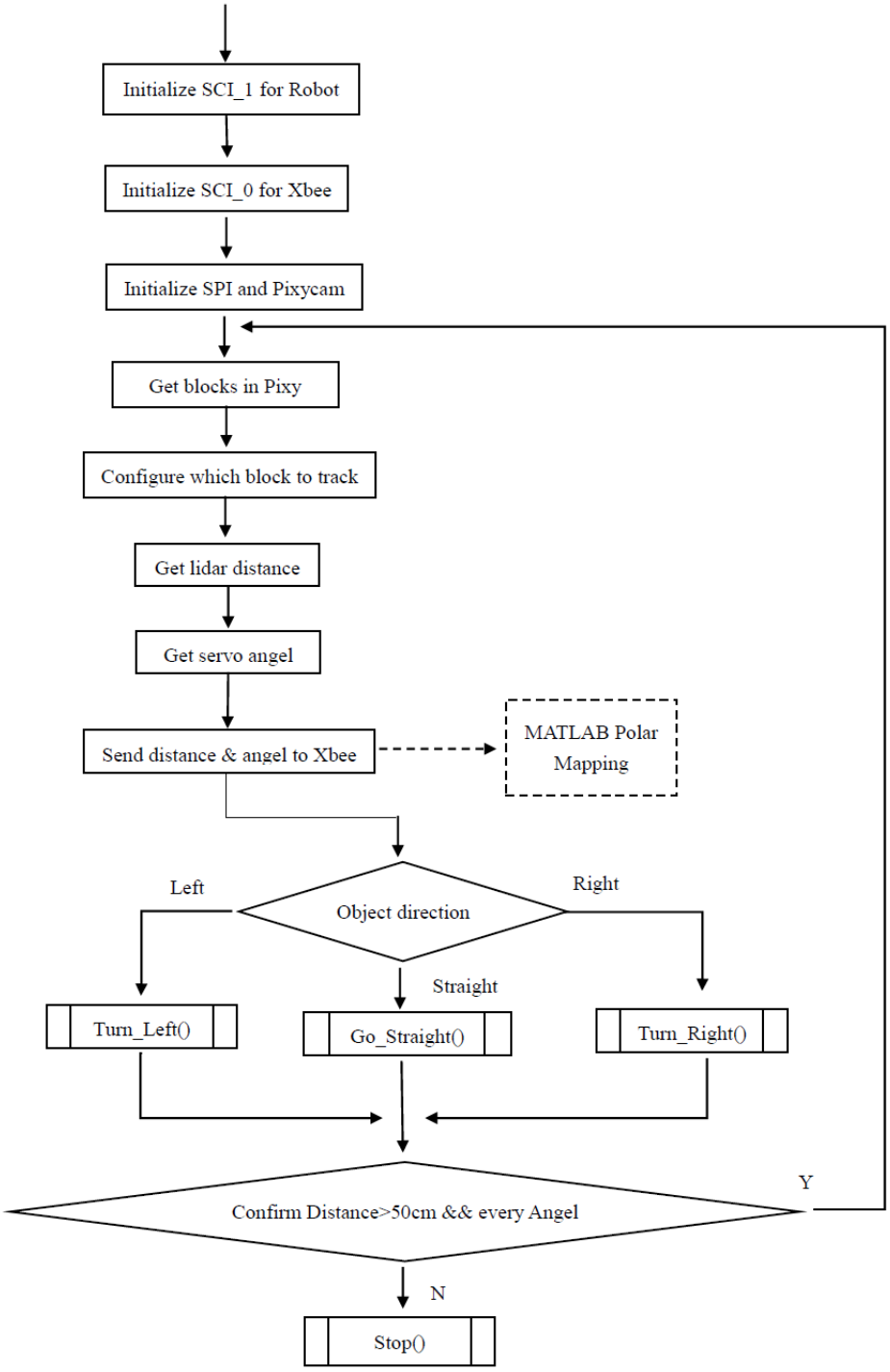


Figure K: Program Flowchart

Note: this diagram is not 100% accurate and may not reflect some design statements, as software design is still underway.

Timing Analysis

After performing some rough calculations, we found that the most time-expensive component of our program was waiting for PWM signals from our LIDAR. This is due to the fact that every PWM Pulse occurs at intervals in excess of 1000us, and each pulse could be up to 5000us (500cm at 10us/1cm). Considering that each command that our 68HS12 will perform will take 3 to 10us, the pixy cam data collect segment and control algorithm together contribute a fraction of this. However, if we assume every LIDAR data capture takes 5000us and our main routine takes 5000us (which is a very, very long time) even without interrupts our program will still be able to run at 10ms per loop. Overall, 100 times a second should be perfectly sufficient for our pixy camera, which is our most demanding component. Although the interrupts would have allowed us to skip these huge waiting periods, it was ultimately unnecessary.

The primary loop we run consists of our 'data get' – we update the Pixy Cam, get LIDAR Data (which takes a long time) and get our servo data. It should be noted that, as soon as the LIDAR data is collected, it is immediately integrated into our environment data pool and sent to our base station. Lastly, our system determines what direction it should walk in and if it should try to strafe in a particular direction.

Control Algorithm

This consists of three primary parts: data simplification, heading decision, and strafing decision. Each of these parts occurs at a different time in the program.

The data simplification occurs at the time that a PWM pulse is received from the LIDAR. Essentially, the robot will determine what strafing direction the pulse came from using the server angle and how much of a hazard it represents using the LIDAR distance. The closer and object is, the more of a hazard it represents. As long as at least one data point represents a significant enough hazard, the robot will choose not to move in that direction later. Hazard data is stored in the strafing quadrant for later.

The next segment is heading decision – based on the current orientation of the Pixy cam, the robot will choose to move in a particular direction. If the pixy cam doesn't have a target, a counter will start, and the robot will either choose to rotate or walk in a particular direction.

Finally, the robot will use the strafing quadrants to decide if it's safe to walk – first, it will check that no quadrant has an extremely hazardous object in it. If any do, it indicates that there is an object too close to the robot for it to move safely, and the robot will freeze – the heading direction is suppressed. Otherwise, the robot will start by checking the direction the heading is in currently: if it is safe, the robot will move in that direction. If a quadrant has recently had a hazardous object, the robot will instead choose to strafe to the quadrant to the left of the current one – if that one is occupied, the quadrant to the right, then the left of the left, until either the robot moves or no quadrant is available to move to.

Possible Improvements

Overall, although I(John) have many complaints about the lab and class arrangement, most of them are probably traceable back to me. That being said, I do have one recommendation: improved organization of the online documentation. It wasn't very easy to navigate, and although I agree that this rewards thorough exploration of the documentation, at least put the project specifications at the top of the file as well as with the assignment.

Summary

Overall, this project could have gone smoother, but has come together in a fairly satisfying way. The design completed, it will be interesting to see if we can fully implement our system, or if additional problems present themselves. Based on our current state, this result is tentative at best, but I remain optimistic.

Workload Distribution

Student	Wen	John	Ziyu
<Task>	(% contribution) <Contribution>	=	=
Milestone 1			
Initial Hardware design & implementation	75% specific design, 3D part creation, and assembly	25% Initial conceptual systems layout; 3D software coordination	0%
Chassis Interface	50%	50%	---
Report	8% Checked writing	84% Wrote Report	8% Checked writing
Milestone 2			
Power System	100%	---	(40% - designed a separate power system; see Major Revisions)
Servo Interface	100% Implemented; debugged bad SCI1	(15% - learned content somewhat later)	---
Pixy Camera	100%	---	---
Report	8% Checked Writing	92% Wrote Report	---
Milestone 3			
LIDAR PWM Interface	25% Performed debugging	75% Solved Debugging, provided extended explanation	---
XBee Interface	20% Built HS12 Interface End (putchar())	(5% Participated in explanation and brainstorming)	80% Produced Matlab Code
Report	---	85% Wrote Report	15% Wrote contributing segment
Milestone 4			
Behavior Routine	25% Brainstorming; wrote current simple behaviors *10%	25% Brainstorming *90% *Will probably write/debug final code*	---

	Code is at fault; needs reworking		
Team coordination	20% Provided some major project updates as completed; requested assistance in debugging	35% Set up team chat; ensured functionality btwn members; tried to organize/enact project plan	5% Participated in conversation occasionally
Presentation Preparation	---	35% printed poster; produced some presentation prompts	%15 Assisted hanging method
Poster Design	20% Provided image support; significant design alterations	65% Wrote/arranged text; Provided technical aesthetic support; significant design impact	15% Provided image support
Report Writing	5% Contributed to report design; provided example reports; participated in some brainstorming; Image support Proofread	80% Wrote. Ya'know. As one does.	15% Provided image support; wrote Xbee section of report; wrote power supply section of report; proofread

***These components remain incomplete at this time. The italic number is the anticipated result. The normal number is the current result.**

Table A: Workload Distributions

Conclusions

Although this project is incomplete (all modules are functional; it has not yet been integrated), the hardware deadline has not yet been reached (report deadline before hardware deadline? Go figure.). We are confident that we will have enough time to bring our implementation up to par with the design covered in this report.

Beyond direct implementation and design skills, the mistakes and failures of this project are definitely valuable lessons. Even though it would have been ideal for us to not need to address such issues, being aware of a problem and understanding it is one of the most important steps to solving it. In the future, such coordination problems should be solvable in the planning phases of the project. Overall, the project reflected the importance of coordination between team members and the value of modular systems.

References

[1] M. Paulik. Comments and Guidelines for the Final Design report

[2] SPI Block User Guide V02.06 Original Release Date: 21 JAN 2000 Motorola, Inc

[3] Getting Started with XBee RF Modules A Tutorial for BASIC Stamp and Propeller Microcontrollers Version 1.0 by Martin Hebel and George Bricker with Daniel Harris

[4] Connecting Devices to the Pioneer 3 Motor-Power Board From MobileRobots Research and Academic Customer Support

[5] Pixy Pet Robot - Color vision follower Created by Bill Earl

[6] Practical Embedded C Programming for the HCS12 Derrick Klotz Field Applications Engineer

[7] CMUcam4 Command List v1.02 For CMUcam4 v1.02 Firmware

[8] Jason He (graduate mechanical engineering student) for helping with 3D object design.