# PABI: DEVELOPING A NEW ROBOTIC PLATFORM FOR AUTISM THERAPY

A Major Qualifying Project Report:

submitted to the Faculty of:

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for:

Degree of Bachelor of Science by:

Eric Guleksen
Robyn Lindsay
Alexander Woodyard

Date: April 29, 2015

Approved:
Professor Gregory S. Fischer, Advisor
Professor Sonia Chernova, Co-Advisor
Professor Fred Looft, Co-Advisor

# Abstract

Autism Spectrum Disorder is a widespread disorder that affects many children across the world. Children affected by autism tend to exhibit an impaired ability to understand non-verbal social cues and actions. Through the use of Applied Behavioral Analysis (ABA) therapy, improvements in behavior and social outcomes have been observed. We have developed a new, robust, and durable research platform designed to interact with children through basic ABA therapy in order to test the effectiveness of robots in autism therapy. This platform is designed to be able to log therapy sessions while interacting with the child in an innovative way through multiple degrees of freedom. The platform is also designed to be expandable by future researchers with the ability to integrate both additional actuators and sensors. Lastly, the entire structure is modular in its construction, meaning entire modules can be removed and added in the future with minimal effort.

# Acknowledgements

Over the course of this project we received the help of numerous individuals without whom this project would not have succeeded. We would like to recognize their individuals for their assistance throughout all stat

- Joe St. Germain for his continuous wisdom and inspiration of designs, as well as for reviewing numerous revisions of our work
- Mike Delph and Chris Nycz for reviewing our work and providing some insight into shortcomings of previous projects
- Dr. Laurie Dickstein-Fischer for providing the team with clinical insight
- Our advisors for continually pushing us to new heights

# Table of Contents

# Table of Figures

# Table of Tables

# Executive Summary

This report provides an overview of the rebuild of the Penguin for Autism Intervention (PABI) research platform. This older platform is a result of years of previous Major Qualifying Projects hosted at WPI. These platforms were developed with the goal of facilitating applied behavioral analysis therapy (ABA therapy) with a child. After analysis on the previous revisions of PABI, we quickly determined that the platform required an update to the physical structure, its components, and its code base. The rebuild still focused on being used as an ABA therapy facilitator. Each team member developed either the mechanical, electrical, or software elements in order to deeply analyze each section in the planning phases. We include in our analysis a full CAD model and rendering, Altium custom designed PCB boards with the specifications of each device on it, and an overall analysis and design of the full PABI system constructed. Our work focused on implementing the software architecture and interaction, the PCB design, and all physical systems of the robot. At the end of the project, the team produced a viable research platform for teams in the future to expand functionality to PABI.

The report describes the research platform developed at the end of the project, as well as the design process. Listed below are some major accomplishments that the team completed.

- A working API for further developing PABI with a barebones implementation.
- Design and assembly of many different electrical boards which deliver power and read data throughout the robot effectively.
- Construction of a new robot, including working cable driven wings and redesigned head system.

The team also highlights the areas where improvement is needed, and what was worked around for the next team to use. Some of the recommendations are listed below

- Improving the outer appearance of PABI by integrating a layer of stuffing into the final outer layer of cloth
- The use of a simpler microprocessor that has a similar footprint and feature set as the one chosen.
- Supporting files that can allow PABI to execute some predetermined routine

This report does not focus on a complete system integration of the PABI robot that would be ready for clinical trials. At the completion of this project, an additional project was underway to implement the application-level software required to create a system ready for clinical use in therapy sessions.

# 1 Introduction

Autism Spectrum Disorder affects one in 68 children today (Odom et al., 2014). Due to the decreased social performance resulting from autism, children with autism often experience underdeveloped speech capabilities, substantially impairing their ability to socialize with their peers. In addition, children affected with autism often struggle to observe non-verbal socializations that are essential to modern social interactions. Autism is often treated with intensive sessions of one-on-one therapy where a professional therapist works to teach a child with autism basic skills ranging from color recognition and matching, to basic language skills. This Applied Behavioral Analysis (ABA) therapy is most effective when conducted for 40 hours a week or more (Lovaas, 1987; Simpson, 2001).

We created an animatronic penguin that could conduct ABA therapy autonomously while logging essential therapy data to be read by a professional therapist. The intent of this was so a professional could tailor the therapy to the individual child while not being at the session. We created an extensible research platform that can evolve over time as the symptoms presented by the child change. This revision was built upon the work done in previous years by creating an animatronic penguin that is robust in its construction and has a robust software base that was extendible and maintainable (Dickstein-Fischer et al., 2011). We developed several key features and systems including a communication framework, low level control system, and a physical system which included mechanisms to actuate 12 degrees of freedom (DoF).

A network framework was developed to manage the communication between the various devices required for the system operation. Through the use of a framework called LCM, a main computer, a sub-computer, and a user interface tablet were able to communicate seamlessly. The network communication protocol was also extensible so that in future work additional items can be transmitted and received through the network with minimal effort. We developed a software suite that was able to track faces and log various therapy events for review by a therapist after a therapy session. Using openCV 2.49 and 720p webcams, we were able to accurately and consistently track a face. We were also able to save video streams and responses from the patient as part of the logging feature.

The electrical system provided the link between the higher level software which created motion commands and the physical robot. A central microprocessor provided a way to interpret commands sent form the higher level software into actions that could be sent to the motors. In addition, we developed numerous custom PCBs in order to provide a way to layout all of the various components required in a compact way that was also conducive to being easily mounted in the final robot.

The physical robot is designed to facilitate ABA therapy by moving numerous axes. The eyes and head could move so that face tracking was possible. The wings moved through the use of a two axis continuum manipulator that provided the structure of each wing. The overall structure centered around an aluminum tube in order to provide a strong, lightweight frame that would stand up to the rigors of interacting with a child. In order to create a self-contained system, we placed all of the computing components as well as the PCBs required to control the system inside the robot. The only item not within the system itself was the tablet, which had to be located outside of the root in order for the child to interact with it.

# 2 Background

## 2.1 Autism Spectrum Disorder

ASD is a neurodevelopmental lifelong disability defined by diagnostic criteria which include deficits in social communication and social interaction and restricted, repetitive patterns of behavior, interests, or activities (American Psychiatric Association, 2013). In recent years, the reported frequency has been approximately 1 in 68 children (Brugha et al. 2011; Odom, S. L. et. al., 2014). Individuals with ASD are often identified between the ages of 12 and 30 months at routine checkups, where they exhibit symptoms such as delayed speech development, an inability to focus on the faces of other individuals, or other similar deficiencies in social skills (American Psychiatric Association, 2013; Leve, Mandell, & Schultz, 2009).

### 2.1.1 Diagnosis of Autism

As of 2013, there are three distinct levels of severity of ASD as determined by the American Psychiatric Association in *The Diagnostic and Statistical Manual of Mental Disorders* (5th ed.; DSM–5; American Psychiatric Association, 2013): mild, moderate, and severe. They are characterized by the amount of supportive care required to compensate for the deficiencies in social and communication skills. For mild cases of ASD, deficits in social communication can cause impairments, but will not prohibit the child from socially interacting with other individuals. For moderate cases of ASD, the subject exhibits deficits in verbal and non-verbal communication skills. Often these children rarely initiate social interaction and exhibit reduced responses to social overtures. Severe cases of ASD often exhibit unintelligible speech and almost no initiative towards interacting with others. They require substantial support to change their behavior and have great difficulty with changes in routine (American Psychiatric Association, 2013).

### 2.1.2 Treatment of Autism

These therapies are often highly structured and focus on reproducing problem behaviors (Lovaas, 1987). After recreating the problem behaviors, the therapy then changes focus on teaching the patent how to learn in a traditional setting and how to socialize with other individuals. Often this therapy is a joint effort between the parents and the child's school system, where the school will work with the parent and the established therapy program to blend education and therapy for a child. One common type of therapy is adaptive behavior analysis (ABA) therapy, where the caregiver works individually with the child to actively modify their behavior over time (Volkmar & Pauls, 2003).

Applied Behavior Analysis (ABA) is a treatment technique where clinicians work to condition subjects toward a behavior that might otherwise not present itself (Granpeesheh, Tarbox, & Dixon, 2009). For children with ASD, this often involves teaching basic social skills as well as elementary concepts such as colors, matching, and basic language skills. In order to be effective, it has been shown that a child must be exposed to ABA therapy for at least 40 hours a week (Lovaas, 1987; Simpson, 2001). Therapy sessions are often focused on imprinting a single skill and measuring how effective previous sessions have been in doing so. A specific approach often used for this is discrete trial training (DTT)

(Lerman et al., 2013). This approach allows the clinician to target specific problem behaviors and provide treatments that will work to eliminate them. Furthermore, DTT is simple enough that it allows parents or other non-trained individuals to facilitate the treatment (Simpson, 2001).

## 2.2 Existing Robots in Autism Therapy

One method that has recently emerged for performing ABA therapy with children is through the use of robots. It is believed that children with autism may be able to relate better to robots than to people, and many different studies have been done to investigate this. The following sections describe a number of past robots that have been designed to help spur social interactions in autistic children. Many of the options that currently exist are somewhat humanoid in nature and focus on having expressive faces. Several also have moveable limbs for added communication. Most of these robots are not fully autonomous, and this is one of the main areas that PABI aims to improve upon. The following sections explore some examples of these alternative therapy robots in more detail.

### 2.2.1 Paro

**Figure 1: PARO Robot (Kidd, Taggart, & Turkle, 2006)**

Paro is a robot resembling a baby seal that was developed with the intention of being used with the elderly, hospitalized children, and autistic children to promote social interaction with the robot and other people (Robins, Dautenhahn, Boekhorst, & Billard, 2005). Paro is able to sense touch, recognize speech, make a limited range of sounds, and move its head and front flippers, but does not have any eye movement. In addition to being designed to promote social interaction, Paro was also designed to encourage subjects to form an emotional bond with the robot and a study was performed at nursing homes to determine the effect of Paro on the residents. During the experiment, it was found that the visually appealing design of the robot had a very positive effect on the initial interaction with the robot, as it was non-threatening enough that the elderly felt comfortable playing with it. It was also found that the residents enjoyed being able to touch the robot and have it interact with them, suggesting that making the robot visually appealing and easily approachable helped it gain popularity with the residents (Kidd, Taggart, & Turkle, 2006). Paro was able to promote social interaction from elderly test subjects suffering from a variety of psychological disorders and improve social interaction. (Kidd, Taggart, & Turkle, 2006).

### 2.2.2 Robota

**Figure 2: Robota (Robins, Dautenhahn, Boekhorst, & Billard, 2005)**

Robota is a small humanoid robot with the similarity of a young girl and has shown to promote social interaction with autistic children. Her movement capabilities include four 1 degree of freedom (DOF) limbs that can move up and down and could respond to touch by sensing passive movement of its limbs. Robota has capabilities for vision tracking and machine learning, but neither of these functionalities were implemented in any tests with children. During tests the Wizard-of-Oz technique was used, where an operator controlled the robot's movements from an unseen location. Overall, the tests proved that the robot was not as useful as desired. For instance, interaction with the robot depended on a teacher helping initiate contact, and the robot would often not detect the child's motions correctly. The tests were rated using the criteria of eye gaze, touch, imitation, and being near the robot. Touch and imitation were both rated very poorly, and eye gaze and proximity to robot were only slightly better, with a slight upwards trend in eye contact at the end of trials and one child interacting with the robot on their own. An interesting phenomenon that these tests also showed was that the children were more willing to interact with the robot when it was in plain clothing and didn't have a face as opposed to when it had the likeness of a young girl. (Robins, Dautenhahn, Boekhorst, & Billard, 2005) Overall, despite being interactive, it was found that Robota was not particularly effective in promoting social interaction in children with autism, as there were no real improvements in any form of social interaction.

### 2.2.3 NAO


**Figure 3: NAO (Shamsuddin et al., 2012)**

The NAO robot is a small humanoid robot capable of blinking, speaking, walking, and moving its arms and head. It is made by Aldebaran robotics and has been used in a couple of tests with autistic children. The researchers deemed a humanoid form would be ideal so that children could copy its movements more easily. In order to blink, the robot flashes LEDs located behind its eyes instead of using eyelids. One emotion NAO can display is fear, which can be alleviated by gently touching NAO's head. During trials, NAO had positive results with autistic children, with showing significant improvement while interacting with it, primarily in an increase in eye contact. It primarily improved stereotyped behavior and communication skills, and also improved social interaction to a lesser degree. The features of the robot are very flat overall in order to be more pleasing to autistic children. (Shamsuddin et al., 2012) One of the major results of this study was that the novelty of the robot compared to humans made it more appealing to the children with autism, suggesting that the robotic features make it more relatable to the children.

### 2.2.4 Keepon

**Figure 4: Keepon (Kozima, Nakagawa, & Yaduda, 2005)**

The Keepon is a small robot created by researchers in Japan for use in autism therapy. The robot is small and yellow with the overall shape of two balls on top of each other like a snowman. It is deformable so that the children interacting with it can squeeze it and play with it without breaking it, and the robot can bounce and tilt in a variety of ways to show different emotions. It also has a seven hour battery life so that it can interact with children for long periods of time without having to be connected to anything else. The primary way that the robot shows emotion is by tilting its body in different ways to indicate happiness, curiosity, sadness, and other emotions. The robot was operated using the Wizard-of-Oz technique during clinical trials, though a secondary automatic mode was also developed. This mode includes functionality for face tracking, color tracking, and algorithms to have the robot look between any faces in its view and occasionally a toy of a predefined color. To do so it makes a cost map which prioritizes faces the most, followed by the toy, followed by other objects. It is also capable of making eye contact using its face tracking software. The Keepon also has a microphone and cameras so that it can hear and see the children. During trials, it was found that the Keepon tended to improve the social interaction skills of the children who interacted with it. The children were typically gradually able to progress to making eye contact with the Keepon, as well as talking to it and touching it. In one notable case, the Keepon facilitated a triadic relationship between a child and another person. (Kozima, Nakagawa, & Yaduda, 2005)

### 2.2.5 Popchilla

**Figure 5: Popchilla App (www.popchillasworld.com)**

The Popchilla is a small creature robot used for autism therapy along with an accompanying tablet app. Popchilla can move its ears and head in order to show emotion, change the color of the LEDs behind its eyes, as well as point vaguely with its arms. The robot is also very soft like a stuffed animal so that children can squeeze it and hug it. This robot works along with an iPad app to teach autistic children daily routines such as brushing ones teeth, making sure that the children learn all about how to perform their day to day activities. Popchilla only has basic functionalities, and is controlled by the app to give positive and negative feedback to the child using the app. The app also can act as a remote control for an outside operator to use the robot in a Wizard-of-Oz style trial (Interbots LLC, 2014).

### 2.2.6 KASPAR

**Figure 6: KASPAR (Wainer, Dautenhahn, Robins, & Amirabdollahian, 2010)**

KASPAR is a robot developed in the UK for social interaction with children. It is the size of a small child with very simplified facial features, including a neutral expression so that the children would be able to interpret the robot as they like. It can also show a range of simplified expressions and move its arms, head, and eyes. The head has 8 degrees of freedom with panning, tilting, blinking, and making simple expressions. The arms have 4 degrees of freedom and are primarily for waving and gesturing. During trials, the robot was programmed to move in certain ways depending on the child's actions during a trial. The main two play scenarios that KASPAR is capable of are turn-taking games and shared-gaze activities, but has other capabilities as well. A series of trials involving the effects of using KASPAR during a cooperative game to encourage cooperative and social interactions with other children or adults. During the trials, KASPAR would play a game with a child the same way a human would play it. It would first play the game with a human, then with KASPAR, and then with another human. An interesting note of this experiment is that KASPAR was completely autonomous for these trials, with no hidden person controlling the robot. The trial concluded that the children were more interested in playing the game with KASPAR than with a human, even after having played with KASPAR. They were also more interested in the final human player than the initial human player, suggesting that KASPAR had a positive effect on their social interaction. Both the children's interest in the game and sociability with other humans improved after a session with KASPAR. (Wainer, Dautenhahn, Robins, & Amirabdollahian, 2010)

### 2.2.7 Infanoid

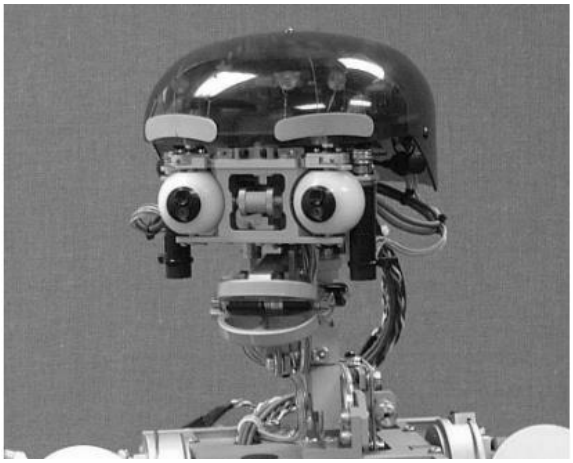


**Figure 7: Infanoid (Kozima & Yano, 2001)**

Infanoid is a humanoid robot that was built with the intent of it being able to be taught simple social interactions by caregivers for eventual use with autistic children. Infanoid has a wide range of facial expressions, with moving eyebrows, eyes, and lips. The concept in the programming of Infanoid is that it would start at the social level of an infant, and be taught the same way an infant would be as to what is socially acceptable, gradually learning to the point that it has the social capabilities of an adult. Infanoid also has cameras in the eyes for face tracking and video logging, and interprets where a person's attention is focused based on the direction of their body, arms, face and gaze. Infanoid has not been tested with autistic children to date. (Kozima & Yano, 2001)

### 2.2.8 Dance Dance Pleo

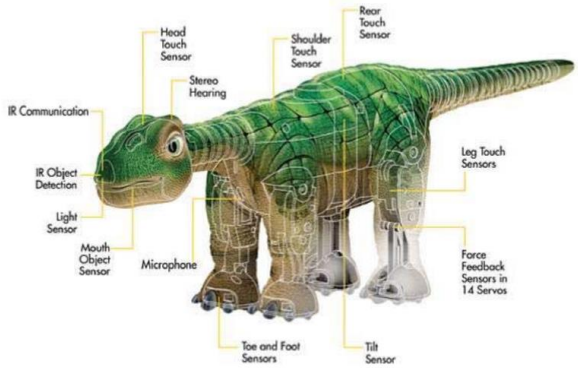



**Figure 8: Dance Dance Pleo (www.pleoworld.com)**

Dance Dance Pleo is a robot resembling a dinosaur made by Innvo Labs Corporation that has been developed such that it can be taught dances and repeat these dances. Dance Dance Pleo has 14 degrees of freedom, and is able to move its legs, torso, neck, eyes, tail, and mouth. It has touch sensors over most of its body, as well as contact switches on the bottom of its feet. It also has a camera in its

nose for object tracking and microphones so it can hear. In order to show emotion, the robot can make happy sounding noises and open its mouth in a smile to show happiness, as well as stomp and make sad noises to show sadness or being upset. The proposed application in autism therapy for Dance Dance Pleo is to have a therapist teach it a dance, and then have the robot dance with different emotions and encourage a child to dance along in a similar fashion. For example, if the robot was dancing happily the child would be expected to dance happily, and if the robot was dancing slowly or sadly the child would as well. Dance Dance Pleo has not been tested in any clinical trials. (Curtis, Shim, Garga, Srinivasan, & Howard, 2011)

### 2.2.9 Dragonbot

**Figure 9: Dragonbots (Kory, Jeong, & Breazeal, 2013)**

Dragonbot is a small robot resembling a stuffed toy that was made by the media lab at the Massachusetts Institute of Technology (Kory, Jeong, & Breazeal, 2013). The robot is controlled by a commercially available smartphone connected to the internet. This allows someone to tele-operate the robot from a remote location and interact with the subject. In addition, the operator can speak and hear through the robot in order to develop the subject's language skills (Kory, Jeong, & Breazeal, 2013). Perhaps the most unique feature of the Dragonbot is the fact that it can be connected to the cloud and adaptively change the way it interacts with the subject in order to better facilitate therapies. Through studies that were conducted through the University of Southern California, it was clear that the robot was an effective method of therapy with the therapist in the room. It was also shown that conducting therapy without the therapist in the room was also effective and able to provide appropriate therapy for the subject (Short et al., 2014).

## 2.3 Differentiating PABI from Existing Solutions

Comparatively, the best way to view how PABI will be different is to first look backwards at progress and then look forward to the future project. No doubt, PABI is a culmination of past projects and robots that have already been done. Cameras in the eye sockets was done by INFANOID, and the

animal shaped idea comes from Paro, but we also see other key mechanical aspects of the robot hailing from Robota with a software model like Popchilla's (Interbots LLC, 2014) (Robins, Dautenhahn, Boekhorst, & Billard, 2005). Where PABI starts to separate from the others is its functionality and general purpose. Out of all of the robots to compare against from above, the ones most similar to PABI to compare against are Popchilla and Robota.

### 2.3.1 PABI vs. Popchilla:

The more recent of the two main PABI competitors is the Popchilla, a chinchilla looking robot that connects to a tablet. While this robot will likely not be used in autism therapy, the system which it interacts with the users and world is startlingly similar to what will be deployed in PABI. The tablet which the Popchilla connects to receives data (in this case, from the tablet) and transmits reactionary procedures to the robot over Wi-Fi. This system allows the tablet and software to be upgraded independent of the robot software itself (Interbots LLC, 2014). The reactions of the robot itself range from simple commands such as a nod, to complex tasks such as singing a song, and displaying other forms of emotion like sadness or inquiry. PABI will be based off of a similar design to this, with the exception of a "stand alone" mode of operation. Since a tablet might not be in the therapy room with PABI, it must be able to determine its own commands and receive data independent from an outside source. Another key feature PABI has over the Popchilla is the ability to log data with and without the tablet. This is key for session playback and detection for professionals that weren't able to observe the therapy session directly. In terms of mechanics, it appears that the Popchilla has more degrees of freedom in its "ears", but lacks in the degrees of freedom in its arms as opposed to PABI. The Popchilla also appears to not have working eyelids, but instead has an interesting LED setup behind the eyeballs in order to simulate open and closed eyes (Interbots LLC, 2014).

### 2.3.2 PABI vs. Robota:

The second, older robot that operates similar to PABI is Robota. While this robot is humanoid looking, it behaves in similar ways to PABI mechanically and also operates similarly when in the therapy session. Robota was placed on a table and mimicked children's movements through raising and lowering arms and legs (each with 1 DOF) (Robins 2005). The robot was said to have an automated mode but it was rarely, if ever, used due to poor sensing and performance. The autonomous mode also required the child to stay in a very confined space, limiting success of the trial further (Robins, Dautenhahn, Boekhorst, & Billard, 2005). The robot instead used remote control for a majority of its trials and testing. PABI learns from these mistakes and senses in a wider arc than Robota due to improved sensors. PABI will also have more degrees of freedom to allow for pointing and better motions on the arms. While PABI does not have movable "legs" the head and eyes greatly make up for the lack of degrees of freedom.

### 2.3.3 PABI vs. Reeti:

Another interesting robot on campus to compare PABI to is the Reeti Robot. Originating from France, Reeti accomplishes similar social interactions with subjects. Also a non-humanoid, Reeti represents a curvy figure with large eyes and pointy antenna ears. The eyes appear to have small cameras in them and are actuated, as well as the ears and head in what appears to be 2 degrees of

freedom. Reeti also utilizes LED lights to assist in showing expressions to the user, as it has no movable body aspects. While PABI does not utilize LEDs to show character or emotion to the user, more body language is used to compensate. It is unknown to compare the software or logging features of Reeti, as no information was available on the matter. What can be confirmed from Reeti is the effective usage of head gestures and cameras in the eyes (Johal, Pesty, & Calvary, 2014).

### 2.3.4 PABI vs. Dance Dance Pleo:

An interesting extreme in the autism therapy is the Dance Dance Pleo robot. As mentioned before, this robot learns dances from the therapist and performs said dances to the patients. This adds a dimension that PABI currently lacks, learning. PABI's sensor bank has plenty of data to interpret and learn motions from, but this project timeframe doesn't have enough time to implement said features. Another interesting feature of Pleo is that it has very unique interfacing with the therapist. In order to learn, Pleo waits for tapping on a movable joint and learns the command one limb at a time. PABI can improve on this by using its extended sensor data to learn multiple movable limbs at one time to save time in the learning process (Curtis, Shim, Gargas, Srinivasan, & Howard, 2011).

### 2.3.5 PABI vs. former versions of PABI:

In addition to looking at other robots used in autism therapy, it was also important to analyze how this revision of PABI was different than the previous versions and iterations of the design. The first immediate design change will be in the overall structure of the robot. The robot will be very compliant with cable driven wings, and will be encased in child friendly foam. This is different from the present iteration in which there was a well-defined rigid shell to protect the electronics, resulting in numerous of servo burnouts. This next revision of PABI will also be deviating from ROS and Linux in order to avoid lack of support for backwards compatibility, and allow future projects to expand on the known platform. Another key improvement from the older versions of PABI is the addition of data logging. Due to the large amount of processing that is predicted, a powerful single board computer with an i7 processor was incorporated into PABI to allow independent hard processing. PABI was, however, be following a style similar to ROS command piping to allow for external entities to control it via Wi-Fi. Following along the same lines as previous PABI projects, there were cameras in each of the two eyes that moved. The wings of PABI also changed from 1 DoF to 2 DoF cable driven compliant wings. These cables, as well as the rest of the moving parts of the robot will be powered by DC motors instead of servos to improve reliability of control surfaces. Using the knowledge from the past and improving in its mistakes allows PABI to take on a new life and a new set of functionality (Dickstein-Fischer, 2011, Levin, 2012, Rivera, 2013).

# 3 Design

## 3.1 Overview

The overall goal of this project was to develop an extensible research platform for autism therapy. This platform was a small, non-threatening, portable robotic system that could interact with a child with autism in order to provide a cost effective alternative to traditional ABA therapy methods. The overall outward appearance was based on a cartoon penguin because penguins are typically viewed as non-threatening and cute.

The design and implementation of the overall system was organized into three areas: mechanical, electrical and software, which are presented in sections 3.2, 0, and 3.4 respectively. Each subsystem underwent serious deliberation before creation. Mechanical designs and decisions are related to the physical components of the robot itself. This ranges from components that the structure is made of, to the various degrees of freedom incorporated into the final system. The electrical section covers the low level control of the motors consisting of component selection, the design of printed circuit boards (PCB) and the embedded software development. Lastly, the software section ranges from programming language evaluation to the protocols used to communicate between devices.

In the mechanical design of PABI, we focused on creating numerous degrees of freedom in order to interact with a child with autism. We constructed the final system by using multiple layers that were connected via a vertical aluminum tube that ran through the center of the system. The spine allowed the outer edges of the system to be soft as well as providing a path through which to run the various wires necessary for the system to function. There were several major subsystems which attached to this spine. The head, including the neck and eyes, was connected to the top of the spine with the wings being connected just below. This created a body cavity where components such as computers and batteries could go. Ultimately, this cavity held the main computer, battery, sub-computer, fan, and printed circuit boards for the microprocessor, voltage regulators, and motor controllers.

The electrical design of PABI focused on designing space efficient printed circuit boards and fast code on the microprocessor in order to ensure that PABI would be able to respond in real time to commands from the main computer. The printed circuit boards were divided into three categories; the main board, the voltage board, and the motor boards. The main board housed the central microprocessor, analog to digital converters, and pulse width modulation generators, as well as breakouts to connect to the voltage board and motor boards. The voltage board housed all of the voltage regulators needed to power the motors and microprocessor, and the motor boards each had a motor control module and a current sensor. The embedded code on the microprocessor was designed to be as efficient as possible so that all of the motors could be controlled at essentially the same time as well as reading all of the potentiometers, encoders, and current sensors.

The software design and implementation of PABI focused on allowing the individual components the most amount of freedom and modularity for future releases. The individual pieces communicated data and status over a network interface in a generalized protocol using libraries that are easy to use and document. The PABI software structure was split into four main sections, each running

on a different device. The more intensive intelligence and processing occurs on the main computer, located inside of PABI running an i7 processor. A tablet is used for child interaction during the therapy sessions and takes in some therapy data that would otherwise be difficult for PABI to take in. The sub-computer, a Raspberry Pi model B+, was located inside of the robot as a translation unit from the higher level commands to specific joint angles and motions that are controlled by the final part of the system, the microprocessor. The microprocessor read in commands from the sub-computer and translated them into real robot movements. A graphical overview of the communication between the various systems of PABI can be seen below in Figure 10.

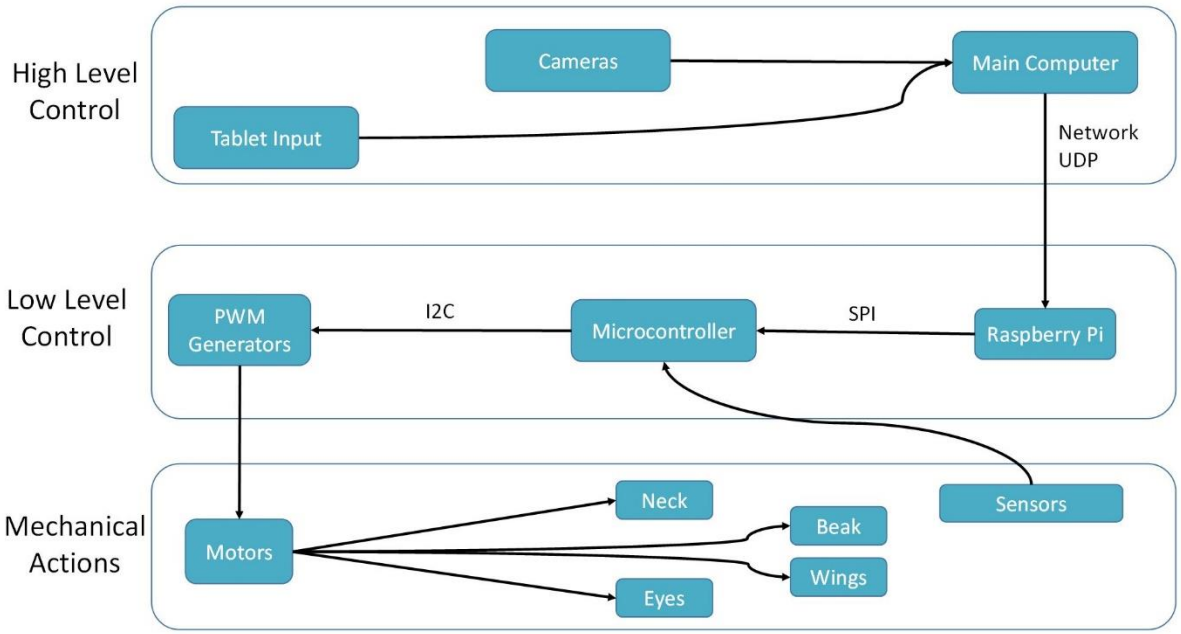



**Figure 10: System overview of PABI**

## 3.2 Mechanical Design

### 3.2.1 Overall Layout

The total size of this system (12 inches wide and 20 inches tall) was similar to the previous versions since the previous version of PABI worked well and was clinically appropriate for this application. The weight bearing components were based on a central spine so that the outer shell of the system could be soft and compliant, encouraging a child to pick up and hold PABI. We mounted the wings and neck to this spine, with the rest of the head being mounted to the neck. The major computing components, including the printed circuit board (PCB) are connected to the spine so as to provide structural rigidity.

### 3.2.2 System Requirements

During our initial design discussions, we came up with a variety of requirements that we felt were critical in order to fully meet our user goals. The mechanical requirements are listed below. They

focused on defining the various axes of motion of the robot as well as some of the necessary clinical requirements various aspects of the robot, listed below:

- Robot will be of robust construction and have multiple degrees of freedom
- The robot will have the following moving parts: beak, two wings, two eyes, two pairs of eyelids, head
    - The beak will open and close in a controlled manner
    - The each wing will have two axis of control
    - Each eye will pan to the left and right independently
    - Each pair of eyelids will open and close around the respective eye independently
    - Both eyes will tilt up and down as a linked system
- Wherever possible, there will be an emphasis on quiet operation
- The robot will be visually appealing to a small child
- The robot will not appear intimidating

Figure 11 and Figure 12 below show renders of the final realization of the mechanical design, both with and without the outer covering.

**Figure 11: Solidworks rendering of entire robot without outer shell, showing internal layout and mechanisms**

**Figure 12: Solidworks rendering of entire robot with outer shell shape**

### 3.2.3 Major Mechanical Subsystems

#### 3.2.3.1 Neck

We divided the robot into three major mechanical subsystems: neck, eyes, and wings. Each provided multiple degrees of freedom as well as an avenue to express emotion in a simple way. The neck provides the structural support for the rest of the components contained within the head. The neck joint was originally planned to have three Degrees of Freedom (DoF) to allow the robot to tilt its head to the right and left, however there were a number of problems with this mechanism shown below in Figure 13. It was very difficult to package an actuator as well as a sensor that provided an absolute position readout within the special confines of the head. In order to keep the overall shape of the head rounded, changes to the design of the neck mechanism were required. We deemed this additional complexity too much of a risk to the overall structural integrity of the system, so we reduced the neck to a two DoF system. The final design of the neck mechanism can be seen in Figure 14 below.

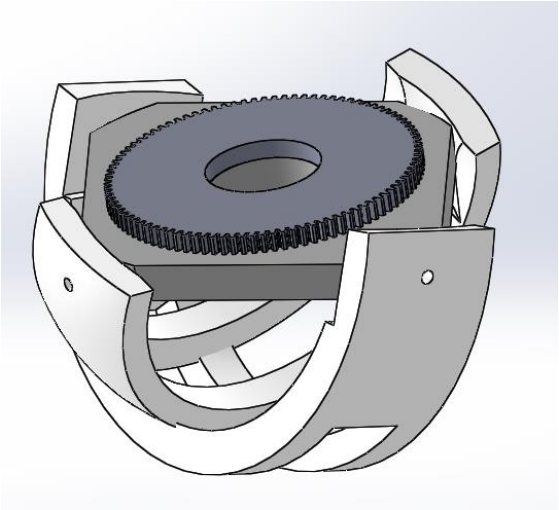


**Figure 13: Initial three DoF neck design**

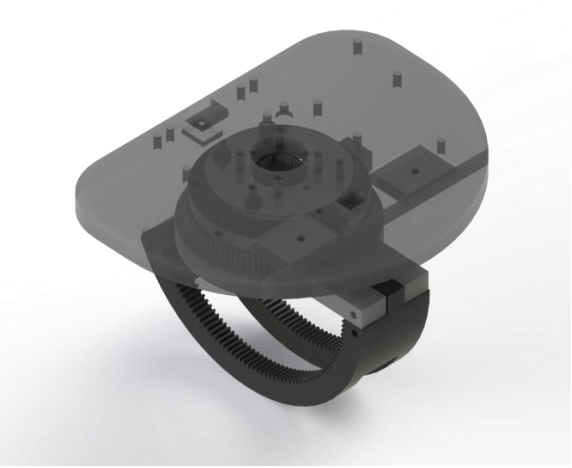


**Figure 14: Final 2 DoF neck design**

This design allowed the head to turn left and right as well as tilt forward and backwards. The neck was designed to pan left and right 90 degrees from center in each direction with the ability to move through 90 degrees of that range within one second. The vertical pan was designed to tilt 45 degrees each direction from horizontal and the ability to move through the full range of motion within one second. The central spine passes through the center of the circular ring, where it is squeezed both above and below by two pieces of plastic as shown in Figure 15. The axis of rotation is in the center of the head similar to humans to provide a reasonable human-like quality to the robot.

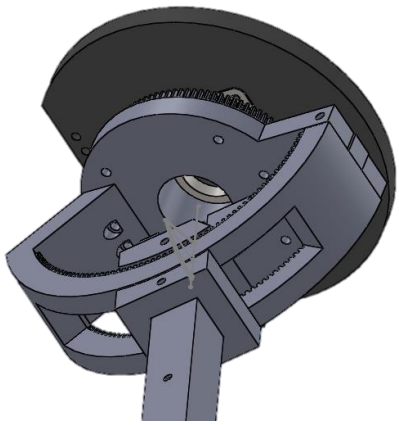


**Figure 15: Neck and spine of PABI showing neck mounted to spine**

### 3.2.3.2 Eyes

We initially intended the eyes to have 3 DoF: independent left/right panning and a tilt motion that was coupled between the eyes. Originally, the eyes were supported on an outer shell of the head (Figure 16) however this was not conducive to the creation of a soft skin to the robot. As a result, having a mechanism that was able to be mounted from the inside of the eye was preferred. This led to a design where each eye was panned left and right via an independent four bar linkage (Figure 17) and the entire assembly was tilted up and down via an additional four bar linkage circled below in Figure 18. Although the original design was contingent on the eyes being supported from the outside of the robot, the final design was supported form the inside of the robot. Similar to human eyes, the eyes on the robot can rotate roughly 35 degrees each direction from center horizontally. The eyes can also rotate vertically up 15 degrees and vertically down 35 degrees. This is roughly half the range of a human's field of view, roughly 60 degrees up and 60 degrees down. While the range of the eyes on the robot is less than the range on a human, it still allows the robot to track a variety of vision targets.

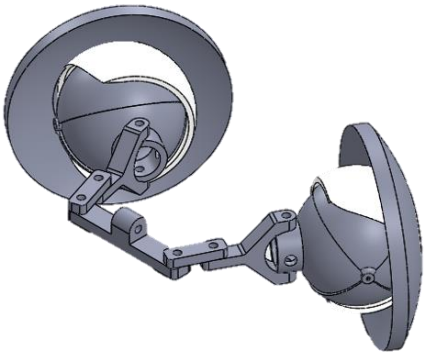


**Figure 16: Initial eye design with support on outside of head**

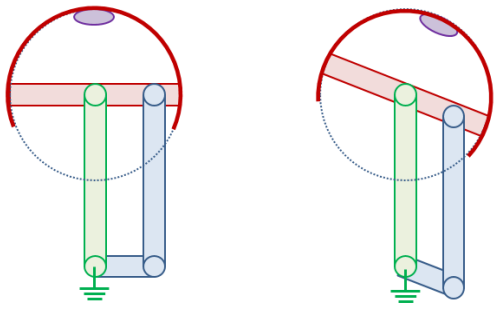


**Figure 17: Sketch of final eye pan design**

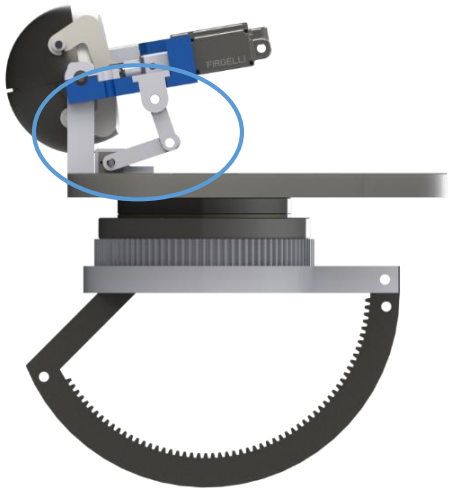


**Figure 18: Four bar mechanism for eye tilt (circled)**

The size and shape of the eyes themselves were loosely based on the eyes of the previous iterations of PABI. The eyes had an outer diameter of 2.5” and contained a plate to mount a small camera (Optimum Orbis 720p camera). Each eye was constructed in two pieces of ABS Plastic that were glued together after the camera was secured to the eye. The two pieces of the eye, as well as the camera that was mounted within the eye can be seen in Figure 19.

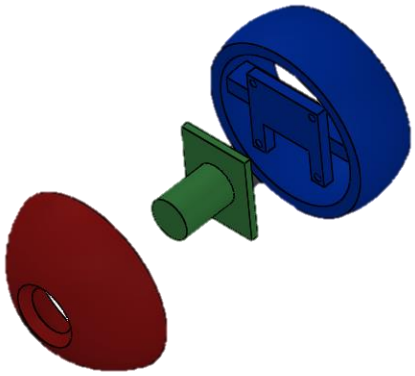


**Figure 19: Exploded view of eye pieces showing enclosed camera (green)**

In order to provide an additional avenue to convey emotion and to add a degree of realism to PABI, two eyelids covered each eye. Two identical slider-crank mechanisms controlled the position of the eyelids, mechanically linking the eyelids over each eye but keeping the eyes independent. The mechanism can be seen in both open and closed configurations in Figure 20 and Figure 21 below.

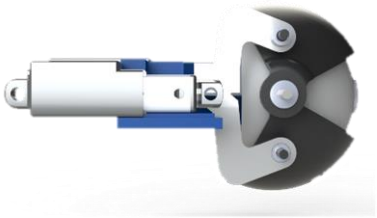


Figure 20: Eyelid Mechanism in the open configuration

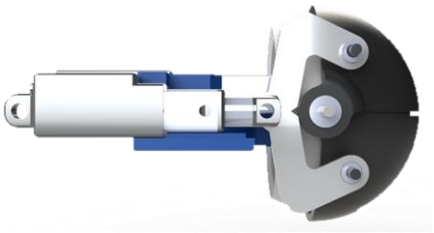


Figure 21: Eyelid Mechanism in the closed configuration

### 3.2.3.3 Beak

The beak was the final mechanical mechanism that was contained within the head. The beak was split into two halves, with the top half fixed in place and the bottom being hinged. The bottom is actuated by a slider crank mechanism that rotates the bottom half of the beak around the hinged edge. This design, shown below Figure 22 mimics a natural penguin in the way the top of the beak is fixed while the bottom of the beak is free to move.

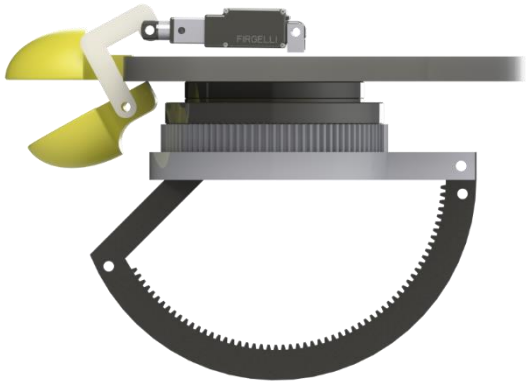


Figure 22: Beak and neck mechanisms

### 3.2.3.4 Wings

The design of the wings for PABI presented a significant challenge. In previous iterations, the wings on PABI were simple, one DoF mechanisms. These mechanisms had limited therapeutic uses, and were more novelty items designed to excite a child. In this iteration of PABI, the wings were intended to bring a new facet of interaction to therapy. In order to accurately point to objects during future therapy sessions, the wings had to be made more robust.

The structure of the wings not only had to be very flexible in order to allow the wings to be able to point at objects, but also durable to stand up to the rigors of being aggressively bent and manipulated by a child. Looking to the medical industry, a nickel-titanium alloy called nitinol, often used in internal medical procedures, fit our requirements.

We determined that a cable driven system would deliver a compliant mechanism that would be able to point and gesture in a way that would greatly enhance the capabilities of PABI. To achieve this design goal, each wing had two axis of control: vertical and horizontal. Each axis had a single cable that

was attached to a spool and motor that could pull the wing in either direction. For example, in the vertical direction, there was a single cable that could pull the structure up or down, depending on which direction the motor turned. Along each wing, there were several locations where the strings were held at a fixed distance from the nitinol running along the center of each wing. By varying the location of these points, we were able to dictate the curves at which the wing would bend. An annotated cross section of the wing can be seen in Figure 23.

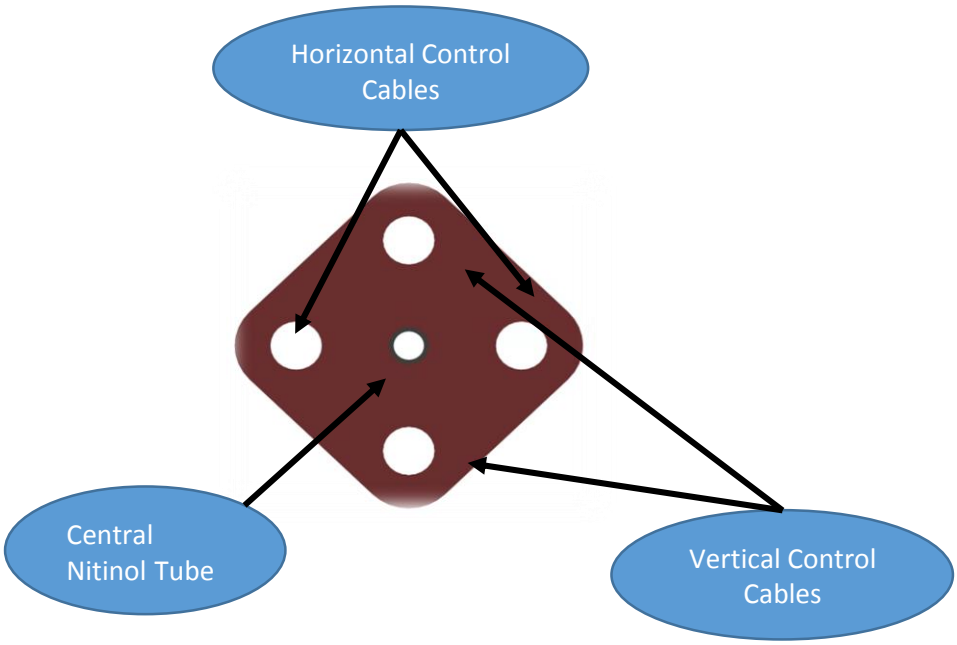



**Figure 23: Annotated cross section of wing**

At first, the wings did not bend as intended since the pieces of acrylic holding the string were rotating in an undesired manner. Instead of staying in a fixed orientation with respect to the nitinol tube, the acrylic pieces would rotate with gravity. This caused the wing to bend to point down during testing, even though the intent was for it to move in order to point up. In order to combat this, it was necessary to glue the pieces of acrylic to the plastic tube that dictated the spacing along the nitinol as well as glue the tip of the wing to the base structure in order to prevent the wing as a whole from rotating. The final wing assembly including the cable used to manipulate the wings can be seen below in Figure 24.

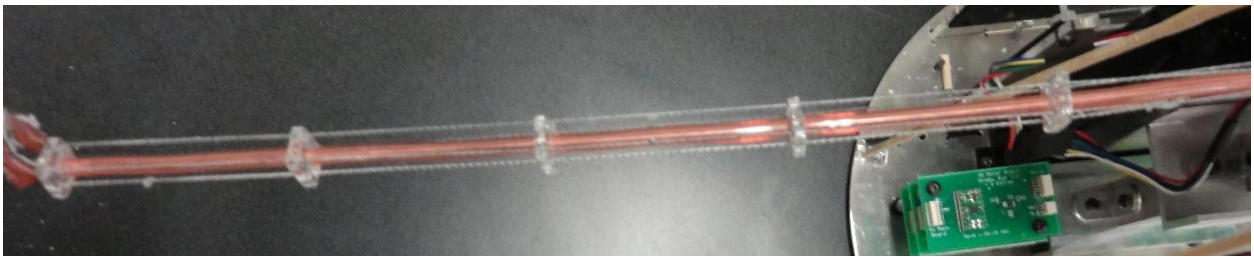


**Figure 24: Picture of assembled wing without external shell components**

### 3.2.4 Internal Layout of Components

There were a number of major components that were required to be inside of PABI so PABI could be a complete, self-contained system. These components included the main computer, sub-computer, microprocessor, a batteries to provide power to the system, a router to facilitate all internal and external network communication, and a speaker to allow PABI to speak to children. In addition, there were numerous PCBs that provided local control of the robot that were required to be inside PABI.

The main computer was placed on one side of the central spine In order to minimize the impact of its high operating temperature. The rest of this side of the internal body cavity was filled with a fan as well as five of the necessary PCBs required for controlling the motors. The empty space left on this side was driven primarily by the need to ensure adequate airflow to ensure that the computer would not overheat.

On the other side of the central spine, components were placed from the inside out based on size and weight. The battery was placed against the central spine due to its weight, with the two largest PCBs mounted against the battery due to their size. The PCB that contained the main processor was mounted on the outside to allow easy access for maintenance and debugging. This arrangement can be seen in Figure 25.

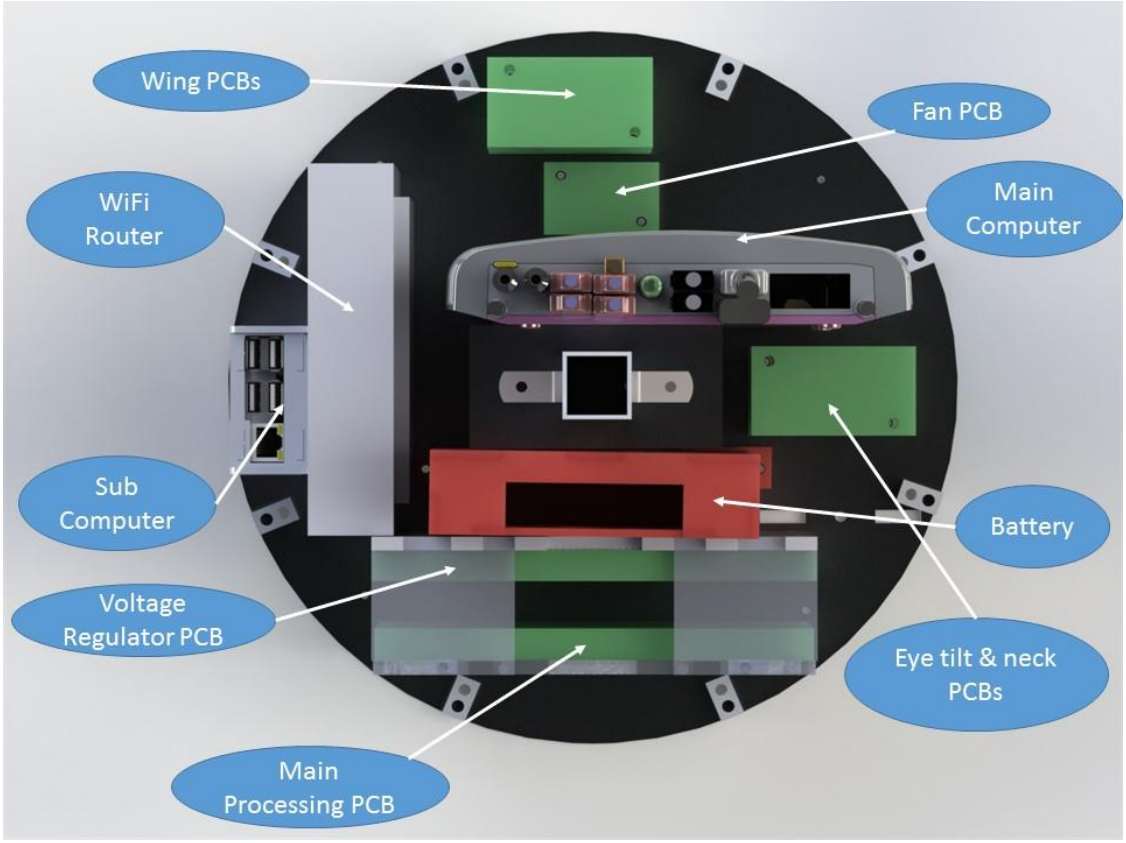



**Figure 25: Solidworks rendering of top-down view of robot base**

We placed the router, sub-computer, and other smaller components around the rest of the larger components wherever they would fit. The router was placed against the ends of the battery and main computer, with the small circuit boards stacked on top of each other where possible and placed on in the remaining flat surfaces the body cavity of the robot.

### 3.2.5 Motor and Sensor Selection

We focused on selecting devices that were robust, quiet, and, as low cost as possible while selecting motors and sensors. For several mechanisms, a linear motion was desirable due to the position curve of the mechanism. Having a linear actuator also allowed us to have a more compact design, which was critical in areas such as the head, where there were many degrees of freedom in a small, confined space. As a result, we looked to small linear actuators for our use. The L12 series of linear actuators by Firgelli Technologies was small and included embedded potentiometers for absolute position sensing. The eye pan actuators were model L12-P-30mm-50:1:12V and the actuators used for the eyelids and beak were model L12-P-10mm-50:1:12V. Each of these linear actuators provided a 2.697 force moving linearly at 0.433 inches/second. This resulted in the eyes being able to move through the full horizontal range in approximately in one second.

For the neck of the robot, it was necessary to have a motor on each axis of rotation. In order to move the actuator from the center of rotation to the outside, we used a gear system on each axis where a pinion gear on a motor would drive a fixed gear on the mechanism itself. For the horizontal rotation of the neck, we used a high power Pololu micro gear motor with a 75:1 gear ratio (Pololu Part Number 2215), providing a maximum of 22 oz-in of torque which was more than adequate to rotate the neck. To tilt the neck vertically, we used a Pololu 37mm gear motor with a 30:1 ratio (Pololu Part Number 1443) providing a maximum of 110 oz-in of torque. In order to make the relative position measured by the encoders mounted on each motor measure absolute position, we included a limit switch on each axis of rotation so that we could home the neck on startup to determine absolute position. Both of these motors were difficult to back drive by hand, but were compliant when driven through the gear system. This behavior was deemed acceptable to use around children.

The wings of the robot required one motor to drive each axis of movement independently, for a total of four motors. The motors were required to generate a large amount of torque in order to bend the nitinol within the wing. Since the motors did not have to turn with any significant speed, a motor with a high gear ratio focused on torque was needed. The Pololu Robotics 25mm metal gear motor with a 172:1 gearbox (Pololu Part Number 2288), which provided 170oz-in of torque was selected for this application. In order to provide absolute position sensing for each axis, we began to explore different types of potentiometers. Since the cable reels would have to make multiple revolutions in a given direction, it was determined that a multi-turn potentiometer was appropriate for this application. In order to have the most robust and flexible sensing system possible, we elected to use 10-turn potentiometers that were press fit directly in the cable reels for simplicity. A summary of each motor, its center position and the effect of each actuator can be seen in Table 1: Summary of motor position information below.

| Motor ID Number (Software Protocol) | Motor Name | Orientation of center/zero position | Positive direction of movement |
|---|---|---|---|
| 4 | Left Eye Pan | Pointing straight ahead | Toward the outside of the robot |
| 10 | Right Eye Pan | Pointing straight ahead | Toward the center of the robot |
| 6 | Left Eyelid | Eyelids open | Closing the eyelids |
| 6 | Right Eyelid | Eyelids open | Closing the eyelids |
| 1 | Eye tilt | Eyes horizontal | Tilting the eyes down |
| 5 | Neck Pan | Pointing straight ahead | Panning the head to the right |
| 7 | Neck Tilt | Head horizontal | Tilting the head up |
| 0 | Beak | Beak closed | Opening the beak |
| 3 | Left Wing Vertical | Wing is straight out | Tilting the wing up |
| 2 | Left Wing Horizontal | Wing is straight out | Panning the wing forward |
| 9 | Right Wing Vertical | Wing is straight out | Tilting the wing up |
| 8 | Right Wing Horizontal | Wing is straight out | Panning the wing forward |

**Table 1: Summary of motor position information**

### 3.2.6 Kinematic Analysis

In order to determine the orientation of the end of the wings and the eyes, we modeled the robot as a series of simultaneous serial manipulators. Each eye and each wing was modeled as a separate, independent collection of links in order to generate a unique vector for each system. Figure 26 below shows a side view of PABI with the shell removed. The labeled dimensions represent the vertical distances between the major components, including the neck's center of rotation.

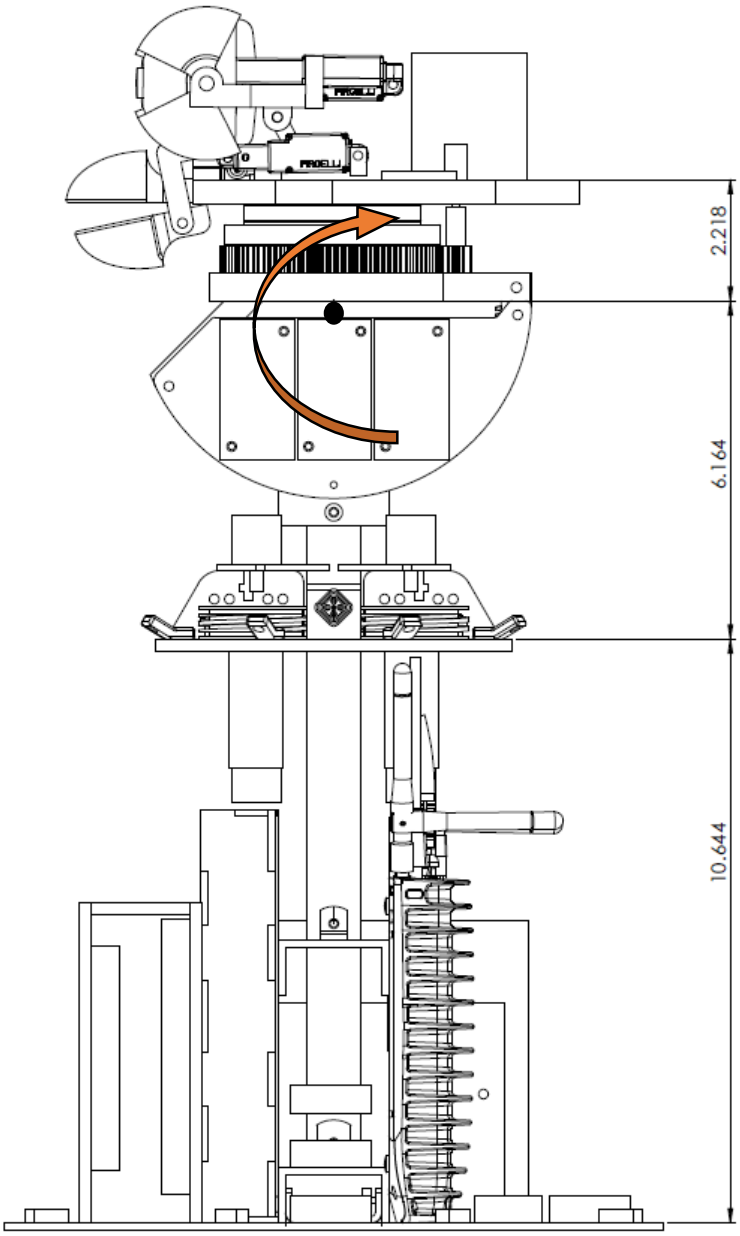



**Figure 26: Dimensioned drawing of entire robot showing dimensions between major subsystems. The arrow indicates the positive direction of neck tilt and the large black dot indicates the center of rotation for the vertical movement of the neck. The positive direction of travel for the neck pan motor is moving the head from left to right.**

While the eyes moved vertically together, they moved horizontally independently of one other, similar to human eyes. Figure 27 and Figure 28 below shows the relation of each eye in relation to the rest of the entire assembly. The positive direction of each eye is indicated by the arrow about the center of rotation for each eye. Figure 29 shows a cutaway of the eyelid mechanism, including the critical dimensions between the linear actuator, the center of rotation for the eyelids, and the lever arm on the eyelid. The relationship between the eye mechanism and the rest of the head is shown in Figure 30. It also includes the critical dimensions for the four bar mechanism that controls the vertical angle of the eye assembly.

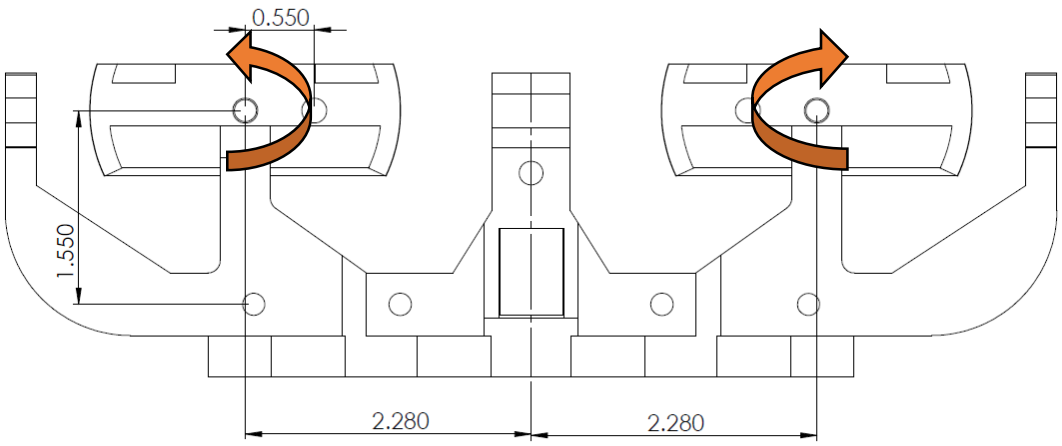



**Figure 27: Dimensioned figure of eye linkage from above with arrows to indicate the positive direction of eye pan rotation.**

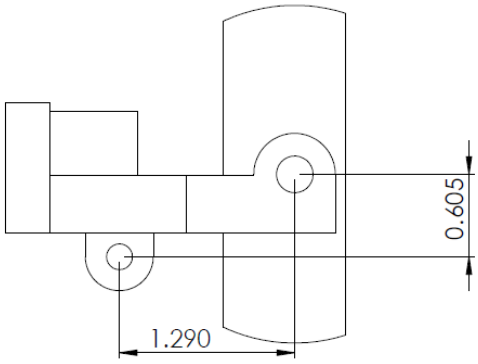



**Figure 28: Side view of eye linkage showing link length for the four bar mechanism which controlled eye tilt**

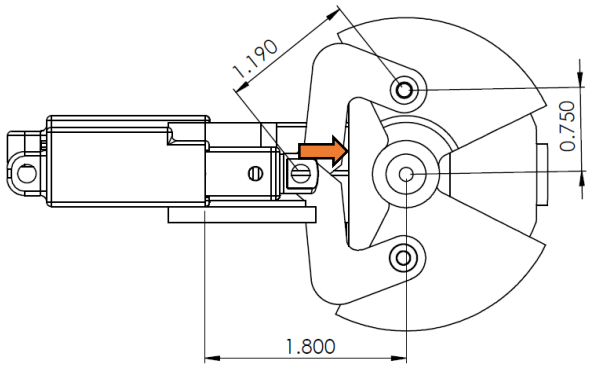



**Figure 29: Dimensioned drawing of eyelid mechanism. Arrow indicates the positive direction of actuating the eyelid motor**

The wings themselves could be modeled as two independent continuum manipulators. Mathematically modeling each of the wings was outside the scope of this project, so in order to move the wings to certain positons, we recorded the extreme positions and interpolated between them. The base of each wing was mounted perpendicular to the front of the robot, three inches from the center. Each motor for the wings was mounted two inches from the center of the robot at 90 degree intervals from each other. The motors are offset 45 degrees from the cardinal axes of the robot as a whole, shown in Figure 31 below.

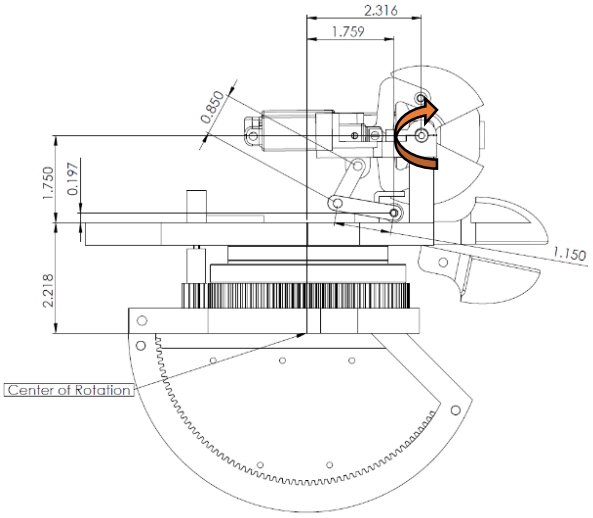



**Figure 30: Dimensioned drawing of head. The arrow indicates the positive direction of travel for the motor which controls the tilt of the neck.**

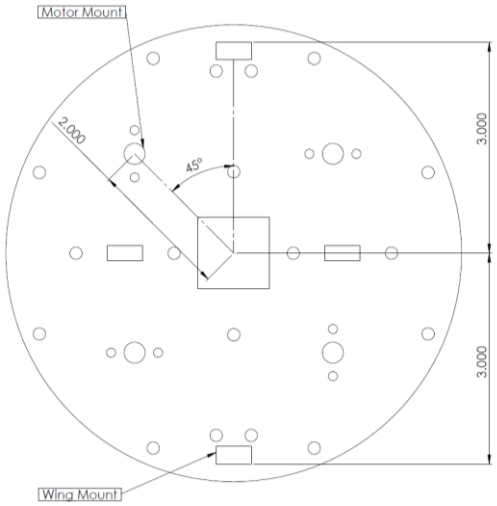



**Figure 31: Dimensioned drawing of the base of the wing module. The wing, which can be represented as a continuum manipulator, begins three inches from the center of the robot.**

## 3.2.7 Production Process

### 3.2.7.1 Prototypes and Iterative Development

After we completed initial designs, we made low cost prototypes of many of the smaller components in order to ensure all the mechanisms worked as intended and that the parts moved as intended. One of the most complicated groups of mechanisms was centered on the eyes. At first there were several clearance issues with the wire associated with the camera. These were able to be solved by eliminating much of the material that supported the camera in order to allow the wire to move easily. Additional space also had to be added between the mount plate for the camera and holes for mounting the eye to the rest of the head to allow for clearances while turning.

The eyelids presented a unique challenge to manufacture, as they were both thin and fragile. At first, there was a lot of friction between the eyelids and the shaft that they rotated about. By adding bushings to the shafts and using flanged bushings where surfaces were sliding against each other, much of this friction was eliminated and the parts were able to move when powered by the appropriate actuator.

The wings were perhaps the most difficult system to manufacture after the design was completed. Originally, the strings that changed the shape of the wing were very close to the central piece of nitinol that ran along the center of each wing. Initial tests proved that this was not an effective method of controlling the wings as the lever arm between the string and the center piece of nitinol was not large enough. By increasing the lever arm, the string was able to exert more torque on the nitinol, bending in and enabling the wing to point and gesture as originally intended.

Initially, the reels that pulled the string on the wings were press fit directly onto the D-shaft of the motor. While this worked for a time, the motors produced so much torque that they rounded out the D shape in the plastic over time, making it impossible for the motors to move the reels. In order to combat this, aluminum shaft adapters were manufactured that would transmit the torque from the motor to the plastic over a larger area without ruining the plastic. This assembly can be seen in Figure 32.

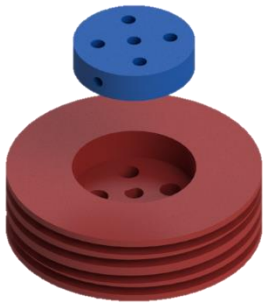


**Figure 32: Exploded perspective of wing cable reel and associated insert**

### 3.2.7.2 Final Production

Nearly every part that made up PABI was a custom shape and could not be purchased from a traditional industrial supplier. As a result, we spent a significant amount of time bringing pieces that were custom designed in Solidworks to life through a variety of production including rapid prototype machines (3D Printer), laser cutter, and a lathe. When possible, parts were cut using a laser cutter due to the lower cost and faster production time. However in order to create some of the unique shapes required, it was necessary to use FDM-based 3D printing machines to create uniquely shaped plastic parts.

### 3.2.7.3 Outer Shell

In order to give PABI the penguin shape required, we covered the internal mechanisms in soft foam that was in turn covered with cloth. This also allowed PABI to be soft and huggable, further enhancing the non-threatening environment that PABI was designed to create. To create the bulbous shape, we used two layers of foam. Around the base of PABI, we placed a cylinder of 2” thick foam that rose to the bottom the wings. To fill in the space between the wings, we used two pieces of thinner 1” foam, one on each half of PABI. These pieces of foam were connected to wooden dowels placed evenly around PABI. Finally, the foam was covered with eight pieces of a soft, stretchable cloth in order to encase PABI in a more natural looking body shape.

To cover the wings, a sleeve was sewn to dictate the final outer shape. Once the wing mechanism was inserted into the sleeve, it was filled with poly-fill, a material often used to stuff pillows, to allow the wing to bend as much as possible with minimal resistance from the stuffing materials used.

The outer shape of the head was realized through the use of 3D printing a shell that would both define the overall shape of the head as well as allowing the external features of PABI such as the eyes

and beak to be visible. The shell was manufactured in four different pieces in order to allow it to be easily installed and removed for service and maintenance of the mechanisms contained within the head. The beak was 3D printed using yellow ABS. Its soft, matte appearance was intended to accurately mimic the visual characteristics of a real beak, so it was not covered in any extra material. The eyes and eyelids were 3D printed using white and black ABS, respectively and were not covered in any additional cloth.

In summary, the mechanical design incorporates custom subsystems for the head, wings, and body of PABI that were synthesized together in order to create a modular and robust final product. The final system had 12 degrees of freedom and was manufactured using a variety of methods, and was covered in a cloth suit to create the appearance of a penguin. The final version of PABI without the cloth and foam as well as with the outer covering can be seen in Figure 33 and Figure 34 respectively.

**Figure 33: Fully integrated and assembled PABI including the outer skin**

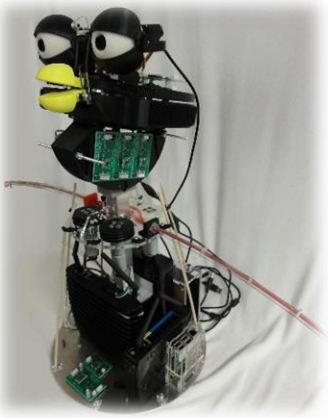


**Figure 34: Fully integrated and assembled PABI without the outer skin**

## 3.3 Electrical Design

The embedded system in PABI was responsible for the low level motor control and sensor reading. In addition it was responsible for communicating information between the main computer and the microprocessor so that the high and low level systems are both aware of what the other is doing. The system consists of a sub-computer (a Raspberry Pi Model B+), a microprocessor, and a number of peripherals that control the motors in PABI. Figure 35 below shows this overview.

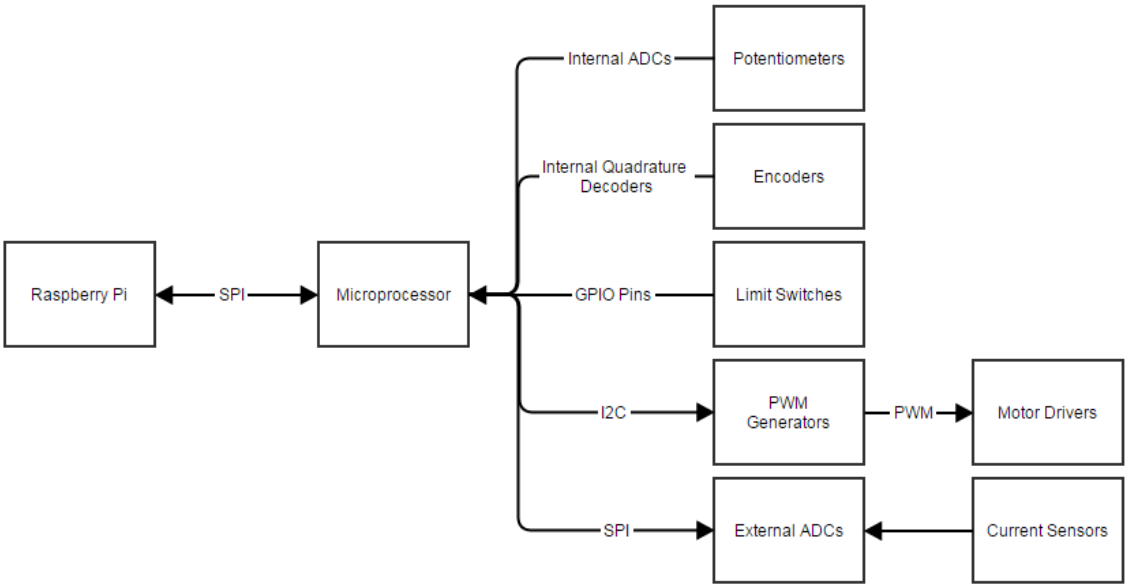



**Figure 35: Embedded System Overview**

## 3.3.1 System Architecture Design

When determining the architecture to use for implementing the embedded system, we decided that a Raspberry Pi B+ could serve as the sub-computer due to its built-in Ethernet port and large amount of documentation. Originally, we proposed to use just the sub-computer for all motor control and sensor readings, but the Raspberry Pi B+ did not have very many SPI ports or ADC channels, which we decided would be problematic for communicating with all of the desired peripherals. The final architecture was a separate microcontroller that would act as a slave to the sub-computer in real time. The sub-computer would receive commands via Ethernet and then transmit them over a serial peripheral interface protocol (SPI), and have the microcontroller act as a master to all other peripherals. As a result, a secondary microcontroller capable of acting both as a master and a slave via SPI was required. This microcontroller also had to be able to control the seventeen motors at once.

## 3.3.2 Component Requirements

### 3.3.2.1 Ethernet Connection to Main Computer

One of the most important requirements for the embedded system in PABI was that it be able to communicate with the main computer throughout operation. We determined that Ethernet was the most effective means of sending large amounts of data between the high level and low level systems. This meant that the embedded system must have an Ethernet port and way of communicating the data it receives to the rest of the embedded system.

### 3.3.2.2 Voltage Regulators

The system was powered by a 12V supply, but some of the motors are 6V motors, so a number of 6V regulators equal to the number of 6V motors was needed. In addition, most chips are powered by 5V logic, so a 5V regulator will be needed as well.

#### 3.3.2.3 Motor Drivers and Pulse Width Modulation Generation

As discussed in section 3.2, PABI uses 13 bidirectional motors, six of which run on 6V and seven of which run on 12V. Therefore, the embedded design required a way of controlling these thirteen motors simultaneously. Additionally, in order to maintain flexibility in the future system, space for four extra motors was included. The two main options for motor control were using either linear motor drivers or H-bridges driven by pulse width modulation (PWM). The use of linear motor drivers was quickly ruled out due to the high amount of heat dissipation. PWM was chosen due to the fact that it is much more efficient in terms of heat. This added the requirement of having enough H-bridges to drive 17 motors at once.

#### 3.3.2.4 Current Sensing

Current sensing was also an important part of PABI's design so that it can detect if any moving parts are being moved by the child at a given time. In order to do this, current sensing was employed on all motors so that a spike in current draw from any specific motor could be detected in case of non-voluntary movement of one of the motors. Therefore, all motors in the system included a current sensing chip.

#### 3.3.2.5 Analog to Digital Converters (ADCs)

PABI's design required a number of analog inputs. Many of these were for the current sensors described in the previous section, while the rest are for potentiometers used on the motors. As discussed in the mechanical design, ten of the motors use potentiometers, and the electrical system must leave the option for four additional motors which may or may not have potentiometers. As a result, fourteen potentiometers and seventeen current sensors, for a total of thirty-one analog inputs, was required.

#### 3.3.2.6 Digital Inputs and Outputs and Quadrature Decoders

PABI also required two digital inputs and a number of digital outputs. The two digital inputs are needed for the limit switch on each of the 2 motors. The digital I\O pins were needed for the SPI and I2C communication protocols. There is also a need for digital inputs for the 2 quadrature decoders.

### 3.3.3 Secondary Embedded Board Design

With these requirements in mind, the components for the board with the microprocessor on it were selected in an iterative process starting with the microcontroller and progressing to the other components.

#### 3.3.3.1 Microcontrollers

##### 3.3.3.1.1 Choosing the Microcontroller

There were two main microcontrollers looked at during the initial development of the electrical system: the ATmega2560 and the AT32UC3C0512C. Both were Atmel microcontrollers because Atmel provides more documentation and support than their competitors.

#### 3.3.3.1.2 **ATmega2560:**

We initially explored the ATmega2560 due to its use in the Arduino Mega, which is often used for motor control. The Atmega2560 is an 8 bit AVR processor that can run at a speed of up to 16MHz. It has one SPI bus with five slave selects, one I2C channel, and four UART channels. It has sixteen 10 bit ADC channels and a maximum of 86 digital I/O pins. We decided that a more powerful processor was needed, as 8 bit processors take a long time to complete most processes, and there is only one SPI bus. This was not ideal, as the system needed to simultaneously act as a master and a slave in an SPI bus. Having a single SPI bus made this impossible to achieve.

#### 3.3.3.1.3 **AT32UC3C0512C:**

The AT32UC3C0512C was looked at because it is another Atmel processor that is more powerful than the ATmega2650. It is a 32 bit AVR processor that can run at a speed of up to 66MHz. It has 2 SPI busses, one of which has three slave selects, and one of which has four. It also has three I2C busses, and five UART busses. The large number of available communication protocols and channels means that it can act as both a master and a slave simultaneously without issue. It has sixteen 12 bit ADC channels and two built in quadrature decoders. This particular model was selected over the 100 pin model so that if more GPIO pins were needed they would be available. More analog channels are required than the sixteen included on this chip, so the use of this chip adds the requirement of 16 external ADC channels. The chip only has 20 dedicated PWM channels, so the system would run much more smoothly using an external PWM generation chip. In addition, this chip requires supply voltages of 5V, 3.3V, and 1.8V, adding a requirement for regulators for each of those voltages as well. The large number of pins on the chip also suggests that having four extra motors is entirely plausible. This brings the total number of analog inputs up to 31 and the number of motors up to eight 6V motors and nine 12V motors. We also later found that this chip had a new type of ADC module that was not very well documented, and found it very difficult to use. The ADC module appears to be a test of a new type of ADC module by Atmel, and there is no support for its use on custom boards. In addition, when looking through Atmel support forums it was found that the ADC module not working properly was a common issue with this chip, so it would be advised to not use this chip again in the future. This is discussed in more detail in the discussion section of this report.

The AT32UC3C0512C was the main microcontroller for this embedded system; however, a number of other components are required in order for it to operate properly. The first of these is an Atmel programmer to use for programming the chip. It was decided to use the JTAG interface for programming, and the AVR Dragon programmer, as it is the simplest and cheapest option available. According to the datasheet, the next part needed is an external crystal. For this project a 16MHz crystal was chosen as it is a good midlevel speed for this chip. A tactile switch is necessary in order to reset the microcontroller when necessary. A number of noise reducing capacitors are also required, including four 100nF capacitors placed near the JTAG voltage supply, the reset switch, the ADC voltage references, and the input voltage to the chip. A 2.2uF and 470pF capacitor were needed close to the VDDIN_33 pin, and a 22pF capacitor is needed on either side of the crystal. A 10kOhm resistor was needed between the 5V supply and the reset switch, and a 47uH inductor was needed near the VDDANA chip. In addition, the 3.3V and 1.8V references are needed, and will be discussed later. Finally, functionality for measuring the

voltage of the 12V battery was needed, which requires a voltage regulator to take the 12V input down to below 3.3V. For this, a 10kOhm and 3.47kOhm resistor were needed. Table 2 below lists the components needed for the AT32UC3C0512C microcontroller to function properly.

| Part | Quantity |
|---|---|
| AT32UC3C0512C microcontroller | 1 |
| AVR Dragon Programmer | 1 |
| 16MHz Crystal Oscillator | 1 |
| Tactile Switch | 1 |
| 100nF Capacitor | 4 |
| 2.2uF Capacitor | 1 |
| 470pF Capacitor | 1 |
| 22pF Capacitor | 2 |
| 10kOhm Resistor | 2 |
| 3.47kOhm Resistor | 1 |

**Table 2: Required Components to Configure AT32UC3C0512C**

### 3.3.3.2 External ADCs

#### 3.3.3.2.1 Choosing an ADC

Due to the large number of analog inputs in the system, the 16 ADC input channels built into the selected microcontroller were not enough to read all of the desired inputs. In addition to the on-board ADC channels15 external ADC channels were necessary to read all 31 analog inputs. The two options looked at for external ADCs were having four 4 channel ADCs and two 8 channel ADCs. Both options were only explored if the ADCs were 12 bit in order to match the resolution of the built-in ADCs. The models of the four channel ADC and the eight channel ADC that we investigated were ADC124S051 and ADS7844 respectively.

The ADC124S051 is a four channel 12 bit analog to digital converter. It can sample at a rates from 200 – 500 kHz and communicates with the main microcontroller over SPI. It has no other GPIO pins necessary to its operation. This means that on the main microcontroller using four of these units would take up four SPI slave select pins and one set of SPI MISO, MOSI, and SCK on the main microcontroller.

The ADS7844 is an eight channel 12 bit analog to digital converter. It can sample at rates up to 200 kHz and communicates with the main microcontroller over SPI. It requires two GPIO pins other than SPI for its operation. This means that on the main microcontroller using two of these would take up two slave select pins and one set of SPI MISO, MOSI, and SCK, and four other GPIOs.

Of these choices, the first choice (ADC124S051) was preferable due to its higher sampling speed and fewer additional GPIO pins, but it also used more SPI slave selects that might also be needed for the PWM generation. It was not decided what the ideal speed would be, but for the purposes of having a fast system, a faster chip was thought to be the better choice. Ultimately, the first option was chosen because the PWM generation chip chosen communicates with the microcontroller using I2C, leaving the four SPI slave selects available for the ADCs.

#### 3.3.3.2.2 Using the ADC124S051 and Additional Components Needed

For this application, the analog inputs needed are the potentiometer values from the 14 motors that use them and the current sense values from the 16 motors that use them. The 14 potentiometer inputs are read by the built in ADC channels on the main microcontroller, but the 16 current sense read of by the external ADCs. This means that four of the ADC124S051 chips are necessary. In addition, according to the datasheet, each of these ADCs require a 1uF capacitor and a 0.1uF capacitor between the VA and GND pins, very close to the chip and in parallel with each other. Table 3 below summarizes the components needed for the ADCs to function properly for this application.

| Part | Quantity |
|---|---|
| ADC124S051 | 4 |
| 1uF Capacitor | 4 |
| 0.1uF Capacitor | 4 |

**Table 3: Required Components to Configure ADC124S051 External ADC Chips**

### 3.3.3.3 Motor Drivers

#### 3.3.3.3.1 Choosing the Motor Drivers

The most important part of the component selection was the H-bridges for controlling the motors. A couple of options were explored, including the TI SN754410 dual H-bridge, the TI DRV8838 motor driver with Pololu breakout board, the TI DRV8835 motor driver with Pololu breakout board the DRV8833 motor driver with Pololu breakout board, and the TI DRV8801 motor driver with Pololu breakout board.

The first of these, the SN754410, is a quadruple half-H driver that can drive two motors by taking in two digital inputs per motor. When a PWM signal is used as one of the inputs, it can be used to drive the motors at varying speeds, and the direction can be switched by switching the input that has the PWM signal on it. This chip can only handle a current of 1A on each motor however, so it is only practical if the two sides are tied together to control a single motor. This is because nearly all of the motors are rated for stall currents of at least 2A, which would easily destroy this chip. This voltage supply range of this chip is 4.5V to 36V, so it would be able to drive both 6V and 12V motors.

The DRV8838 motor driver with Pololu breakout board with an H-bridge that can drive one motor off of one PWM input, and a digital input pin is used in order to control direction. The chip can supply up to 1.7A to the motor continuously with a peak of 1.8A, and can drive motors with voltages from 0V to 11V. This means that this board would only be able to drive the 6V motors. The board from Pololu has built in protection from over current and over temperature.

The DRV8835 with Pololu breakout board is very similar to the DRV8838, but has can drive two motors instead of one. It can drive motors of the same voltages, but can handle a continuous current of only 1.2A and a peak of 1.5A. Also, instead of controlling direction with a digital pin, the chip takes in two PWM inputs per motor. Similar to the other board, there is built in over current and over temperature protection. This board could also only drive the 6V motors.

The DRV8833 with Pololu breakout board is almost identical to the DRV8835 in that it can drive two motors and takes in two PWM inputs per motor. It can power motors from voltages of 2.7V to 10.8V, can supply 1.2A per channel continuously, and can tolerate peak currents up to 2A. As with the other boards, it has built in over current and over temperature protection. This board could also only drive the 6V motors.

The last chip looked at was the DRV8801, which is a higher voltage equivalent of the DRV8838. It can drive one motor using a PWM input and a digital direction input, and has built in over current and over temperature protection. It can handle a continuous draw of 1A and a peak of 2.8A, and has an operating voltage of 8V to 36V. This means that this board could only drive the 12V motors.

Ultimately it was decided to go with the DRV8833 motor driver with Pololu Dual Motor Driver Carrier (part #2130) to drive the 6V motors and DRV8801 motor driver with Pololu Single Brushed DC Motor Driver Carrier (part #2136) to drive the 12V motors. The 12V motors include the two eye pan motors, the two eyelid motors, the beak motor, the neck tilt motor, and the fan motor. The 6V motors include the four wing motors, the eye tilt motor, and the neck pan motor. These were chosen over the SN754410 because of the higher amount of current that they could handle. The DRV8833 was chosen above the DRV8835 because both could only drive the 6V motors, but the DRV8833 could handle a higher current draw. It was also chosen over the DRV8838 because it could handle a higher peak voltage. It was decided that even though it would not always be practical to drive two motors off these, it would still be more useful than the DRV8838 for single motor parts of PABI. In order to drive all of these motors, nine DRV8801 boards and eight DRV8833 boards would be required. Table 4 below is a summarized table of the parts needed in order to drive the motors.

| Part | Quantity |
|---|---|
| DRV8801 Pololu Breakout Board | 9 |
| DRV8833 Pololu Breakout Board | 8 |

**Table 4: Motor Board Quantities**

### 3.3.3.4 PWM Generation

#### 3.3.3.4.1 Choosing the PWM Generator

Since we decided to use PWM generation to control the speed and direction of the motors and since the main microprocessor did not have sufficient dedicated PWM generator channels to drive each motor, it was necessary to choose an external chip to generate PWM signals. The first criteria was the method of communication, as it would not be able to use SPI, since the ADCs used all of the available SPI channels. The main microcontroller had both UART and I2C communication available for use, so a chip using either of those methods for communication would work. The two main chips that were looked at for the PWM generation were the TLC5940 and the PCA9685.

The TLC5940 is a 16 channel PWM generation chip that uses a custom communication protocol that functions in a similar way to SPI. It requires a serial in, serial out, and clock pin as the minimum to control it. The chip was also able to be daisy-chained, meaning that if more than one chip was necessary they could run off of the same control pins. The downside to using this chip was that the communication protocol did not have much documentation, meaning another chip would probably be easier to use.

The PCA9685 is also a 16 channel PWM generation chip, but it communicates using the standard I2C protocol, which is the same as the two wire interface on the main microcontroller. This chip is also daisy-chainable, and also only needs two pins in order to interface with the main device, as opposed to the three that would be needed for the TLC5940. In addition, Adafruit sells a breakout board (Pololu product ID 815) for the chip which adds reverse polarity protection and resistors on each output pin to protect them.

#### 3.3.3.5 Using the PCA9685 with Adafruit Board and Additional Components Needed

The maximum number of motor breakout boards would be eight using the DRV8833 and nine using the DRV8891. The DRV8833 boards require 2 PWM inputs per motor, for a total of 16PWM inputs for the 6V motors. The DRV8801 motors require 1 PWM input per motor, for a total of 9 PWM inputs for the 12V motors. In total, the system requires 25 PWM signals to drive the motors, so two of the 16 channel PCA9685 boards are needed. No additional components are needed for them to function.

#### 3.3.3.6 Current Sensor

##### 3.3.3.6.1 Choosing the Current Sensor

When choosing the best way to do current sensing on the motors, the chip chosen was the LMP8602. This chip was chosen because it can do bidirectional current sensing, has adjustable gain based on the resistors used, and has been used by the team in past projects for current sensing on motors.

##### 3.3.3.6.2 Using the LMP8602 and Additional Parts Needed

Since there are sixteen motors that needed current sensing, sixteen LMP8602s were needed. In addition, each LMP8602 needs a very small current sense resistor between the +IN and –IN terminals, a resistor between the A1 and A2 terminals to determine the gain, and a 100nF capacitor near the Vs terminal. A value of 0.05Ohms was chosen for the current sense resistor, as it is a small enough resistor value to have a tiny voltage drop by which to measure the current. The gain resistor was chosen to be 0Ohms as the internal amplifiers already amplify the voltage to values that can measured on an analog pin. Table 5 below summarizes the components required for current sensing on PABI. .

| Part | Quantity |
|---|---|
| LMP8602 | 16 |
| 0.05Ohm Resistor | 16 |
| 0Ohm Resistor | 16 |
| 100nF Capacitor | 16 |

**Table 5: Summary of Components Required for Current Sensing on PABI**

#### 3.3.3.7 Voltage Regulators

##### 3.3.3.7.1 Choosing the Voltage Regulators

Choosing the voltage regulators was very straightforward, as the voltage regulations needed were a 5V logic supply, a 1.8V reference for the microcontroller, a 3.3V reference for the microcontroller, and a 6V supply for each 6V motor. Each regulator was chosen from Pololu, using buck

converters for the small low current regulators (5V, 1.8V, 3.3V), and switching regulators for the high current draw 6V regulators. Each of these regulators could take in 12V as their supply voltage.

#### 3.3.3.7.2 Using the Voltage Regulators and Additional Parts Needed

We chose to have three 5V regulators so that one could power the logic on the main board, one could power the logic on the 12V motor boards, and one could power the logic on the 6V motor boards. Only one of the 1.8V and 3.3V regulators were needed as they were only used as voltage references for the microcontroller. A separate 6V regulator was needed for each 6V motor, as each regulator could only tolerate a current draw of up to 2A, which was only slightly higher than the peak current draw of the 6V motors. In total, eight 6V regulators were required.

A couple additional parts were needed in order to ensure that the voltage regulators functioned properly, first having a 100nF capacitor between the $V_{in}$ and GND pin of each regulator. Also, a DC barrel jack was needed in order to supply the 12V input to the regulators, as well as a shutoff switch for the regulators as a precautionary measure. For this a two state switch was decided on with a 10kOhm resistor as well to tie it to ground. Table 6 below summarizes the components required for the voltage regulators.

| Part | Quantity |
|---|---|
| Pololu 5V, 500mA Step-Down Voltage Regulator | 3 |
| Pololu 3.3V, 500mA Step-Down Voltage Regulator | 1 |
| Pololu 1.8V, 500mA Step-Down Voltage Regulator | 1 |
| Pololu 6V Step-Up/Step-Down Voltage Regulator | 8 |
| 100nF Capacitor | 13 |
| DC Barrel Jack | 1 |
| 3 Prong Switch | 1 |
| 10kOhm Resistor | 1 |

**Table 6: Summary of Components Required for the Voltage Regulators**

#### 3.3.3.7.3 Choosing the Connections

When choosing what kind of wire connectors would be ideal, it was decided that the connectors should be locking and also able to handle up to 2A of current on each wire. This is because the highest amount of current that should have to go through any one wire is 2A, and having all wires capable of doing so allows for the use of the same wire everywhere. To suit this need, the Molex Pico-SPOX connector system was chosen for meeting all of these requirements.

There was also a need for generic 0.1inch spaced headers to connect the breakout boards to the main boards. Both male and female were used.

#### 3.3.3.7.4 Using the Molex Pico-SPOX Connector System and Additional Parts Needed

For this project, the team decided that all wired connections should use the Molex connectors. Based on the schematics for the boards and the connections between them, it was determined that there would be a need for forty-three 2 pin connectors and housings, ten 3 pin connectors and housings, fifteen 5 pin connectors and housings, fourteen 6 pin connectors and housings, thirty-five 7 pin connectors and housings, and seven 9 pin connectors and housings. In total, this requires for 583 crimp

terminals and a large amount of 24AWG wire. The headers were decided to be horizontal in all connections except the 2 pin connectors, which bridge two boards that are parallel to each other. For the generic 0.1inch spaced connectors, it was estimated that 80 of both male and female headers would be appropriate. We also found it necessary to purchase the Molex Crimp Tool for 1.5mm Pitch Pico-SPOX Female Crimp Terminal in order to assemble the cables. Table 7 lists all connector related components and Table 8 summarizes all electrical components required for PABI. See Appendix A for the lengths and pinouts of each wire.

| **Part** | **Quantity** | **Molex Part Number** |
|---|---|---|
| 2 Pin Molex Pico-SPOX Terminal (vertical) | 43 | 0874370243 |
| 2 Pin Molex Pico-SPOX Housing | 43 | 0874390200 |
| 3 Pin Molex Pico-SPOX Terminal (horizontal) | 10 | 0874380343 |
| 3 Pin Molex Pico-SPOX Housing | 10 | 0874390300 |
| 5 Pin Molex Pico-SPOX Terminal (horizontal) | 15 | 0874380543 |
| 5 Pin Molex Pico-SPOX Housing | 15 | 0874390500 |
| 6 Pin Molex Pico-SPOX Terminal (horizontal) | 14 | 0874380643 |
| 6 Pin Molex Pico-SPOX Housing | 14 | 0874390600 |
| 7 Pin Molex Pico-SPOX Terminal (horizontal) | 35 | 0874380743 |
| 7 Pin Molex Pico-SPOX Housing | 35 | 0874390700 |
| 9 Pin Molex Pico-SPOX Terminal (horizontal) | 7 | 0874380943 |
| 9 Pin Molex Pico-SPOX Housing | 7 | 0874390900 |
| Crimp Terminals | 583 | 87421-0000 |
| 24AWG Wire | 4 Spools | n/a |
| Molex Crimp Tool for 1.5mm Pitch Pico-SPOX Female Crimp Terminal | 1 | 63825-9700 |

**Table 7: Summary of Connector Components**

| Part | Quantity |
|---|---|
| AT32UC3C0512C microcontroller | 1 |
| ADC124S051 | 4 |
| DRV8801 Pololu Breakout Board | 9 |
| DRV8833 Pololu Breakout Board | 8 |
| Adafruit PCA9685 w/ I2C Interface | 2 |
| LMP8602 | 16 |
| AVR Dragon Programmer | 1 |
| 16MHz Crystal Oscillator | 1 |
| Tactile Switch | 1 |
| 100nF Capacitor | 33 |
| 2.2uF Capacitor | 1 |
| 470pF Capacitor | 1 |
| 22pF Capacitor | 2 |
| 1uF Capacitor | 4 |
| 0.1uF Capacitor | 4 |
| 10kOhm Resistor | 3 |
| 3.47kOhm Resistor | 1 |
| 0.05Ohm Resistor | 16 |
| 0Ohm Resistor | 16 |
| Pololu 5V, 500mA Step-Down Voltage Regulator | 3 |
| Pololu 3.3V, 500mA Step-Down Voltage Regulator | 1 |
| Pololu 1.8V, 500mA Step-Down Voltage Regulator | 1 |
| Pololu 6V Step-Up/Step-Down Voltage Regulator | 8 |
| DC Barrel Jack | 1 |
| 3 Prong Switch | 1 |
| 2 Pin Molex Pico-SPOX Terminal (vertical) | 43 |
| 2 Pin Molex Pico-SPOX Housing | 43 |
| 3 Pin Molex Pico-SPOX Terminal (horizontal) | 10 |
| 3 Pin Molex Pico-SPOX Housing | 10 |
| 5 Pin Molex Pico-SPOX Terminal (horizontal) | 15 |
| 5 Pin Molex Pico-SPOX Housing | 15 |
| 6 Pin Molex Pico-SPOX Terminal (horizontal) | 14 |
| 6 Pin Molex Pico-SPOX Housing | 14 |
| 7 Pin Molex Pico-SPOX Terminal (horizontal) | 35 |
| 7 Pin Molex Pico-SPOX Housing | 35 |
| 9 Pin Molex Pico-SPOX Terminal (horizontal) | 7 |
| 9 Pin Molex Pico-SPOX Housing | 7 |
| Crimp Terminals | 583 |
| 24AWG Wire | 4 Spools |
| Molex Crimp Tool for 1.5mm Pitch Pico-SPOX Female Crimp Terminal | 1 |

**Table 8: Summary of All Electrical Components**

### 3.3.4 Designing the PCBs

The PCB design was done using Altium Designer, and a number of revisions were done. The revision history described below includes design reasoning as well as documentation on the changes between revisions. There are seven different board designs, the first of which is the voltage board which houses all of the voltage regulators needed for the electrical portion of PABI. The second is the main board, which houses the main processor and all supplemental components aside from motor drivers, current sensors, and voltage regulators. Lastly, there are five different types of motor boards. There is the 12V Fan Board, which contains a 12V motor driver and no additional circuitry. The rest of the motor boards consist of a motor driver for their respective voltages and a current sensor. There are two other 12V motor board designs, one of which contains wiring for a potentiometer and one of which contains wiring for an encoder and a limit switch. Lastly there are two types of 6V motor boards, one which contains wiring for a potentiometer and one which contains wiring for an encoder and a limit switch.

#### 3.3.4.1 Revision A

Revision A was where the most of the design decisions were made, including the decision to split the board functionality up into a variety of different boards. It was found that there was not enough space inside of the main body of PABI to have all of the components on the same board. In order to split things up, seven different PCBs were designed. The first is the voltage board, which contains the input for the power supply as well as all of the voltage regulation, which then connects to the main board. The voltage board can be seen below in Figure 36.

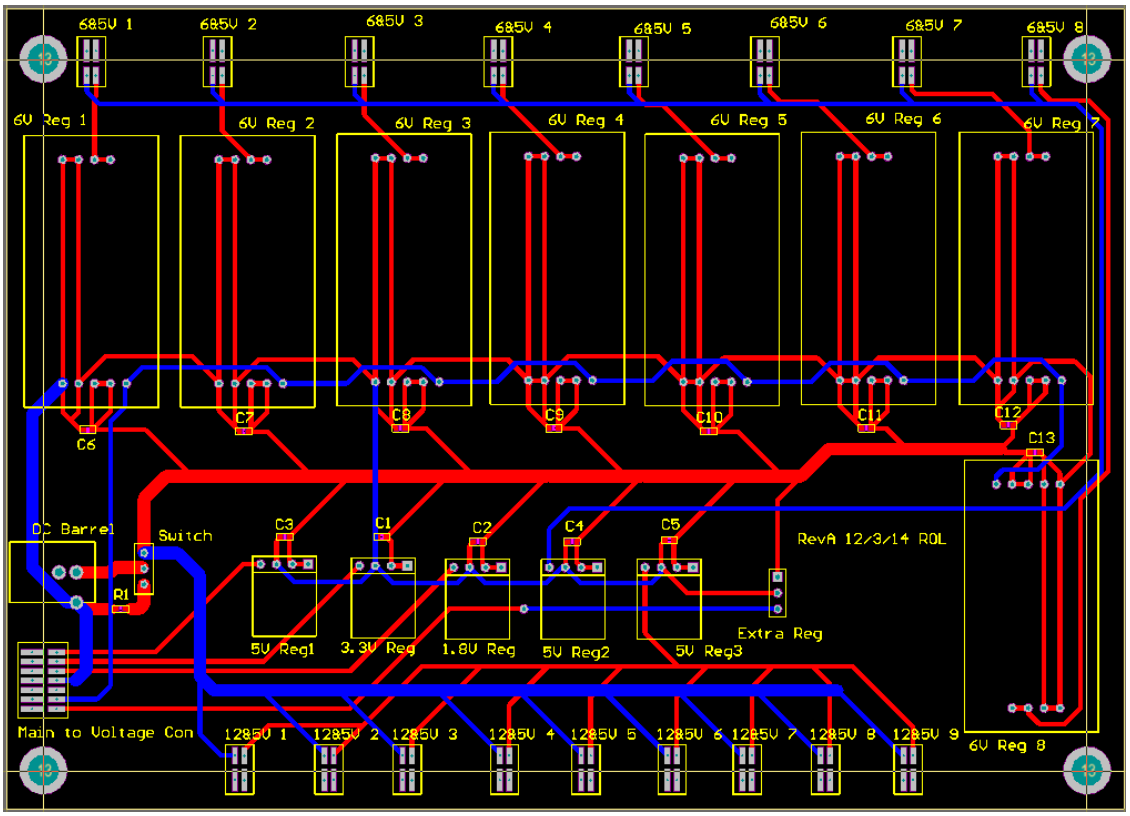



**Figure 36: Voltage Board Revision A**

The main board contains the microprocessor, the external ADCs, the PWM generators, the JTAG interface for programming, the SPI breakout to communicate with the Raspberry Pi, and the breakouts to send and receive information from the motor boards. The main board can be seen below in Figure 37.

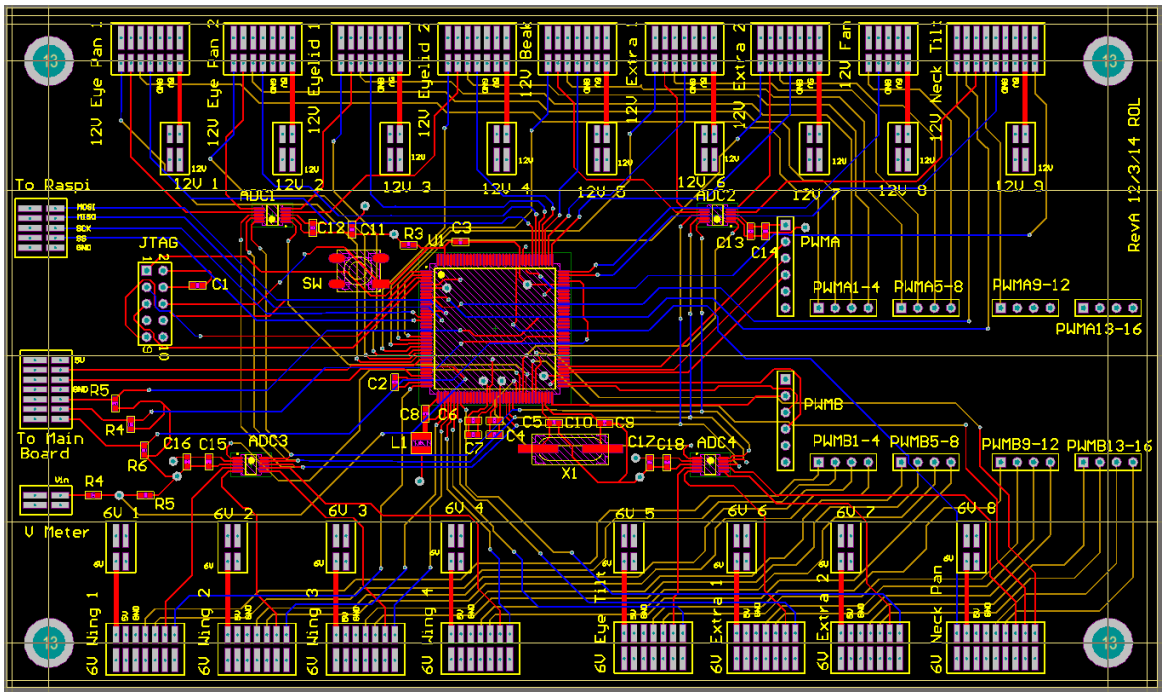



**Figure 37: Main Board Revision A**

The motor boards have five different variations, depending on the type of motor that needs to connect. Differences include the motor controller, which is different for the 6V and 12V motors, as well as the type of sensor used to determine the motor's position. All except the fan motor board include a current sensor. The five types of motor boards can be seen below in Figure 38 through Figure 42.

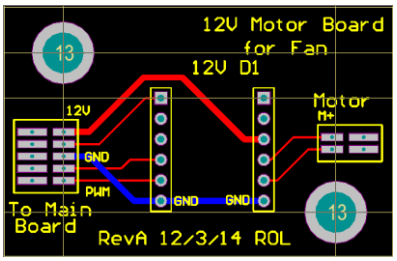



**Figure 38: 12V Fan Board**

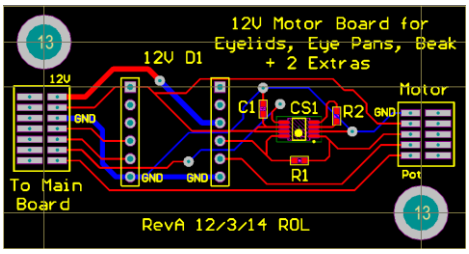



**Figure 40: 12V Eyelids, Eye pans, Beak**

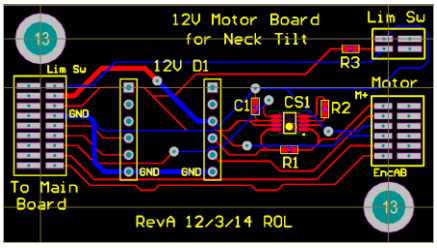



**Figure 39: 12V Neck Tilt Board**

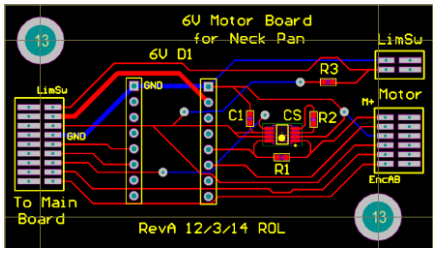



**Figure 41: 6V Neck Pan**

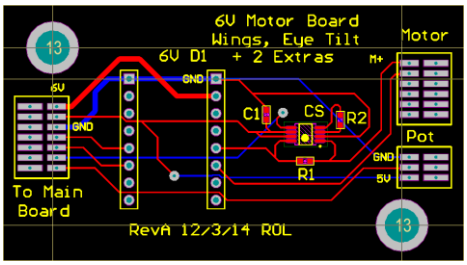



**Figure 42: 6V Wings, Eye Tilts**

In order to save on the cost of printing the small motor board PCBs, the motor boards were arranged into a sheet to have two copies printed.

### 3.3.4.2 Revision B

In Revision B of the boards, a couple of changes were made after testing and finding some flaws with the old boards. For the voltage board, it was found that the footprints for the 6V regulators and for the DC barrel jack were incorrect, and many of the pin holes were too small for the pins to fit through. Revision B of the voltage board can be seen below in Figure 43.

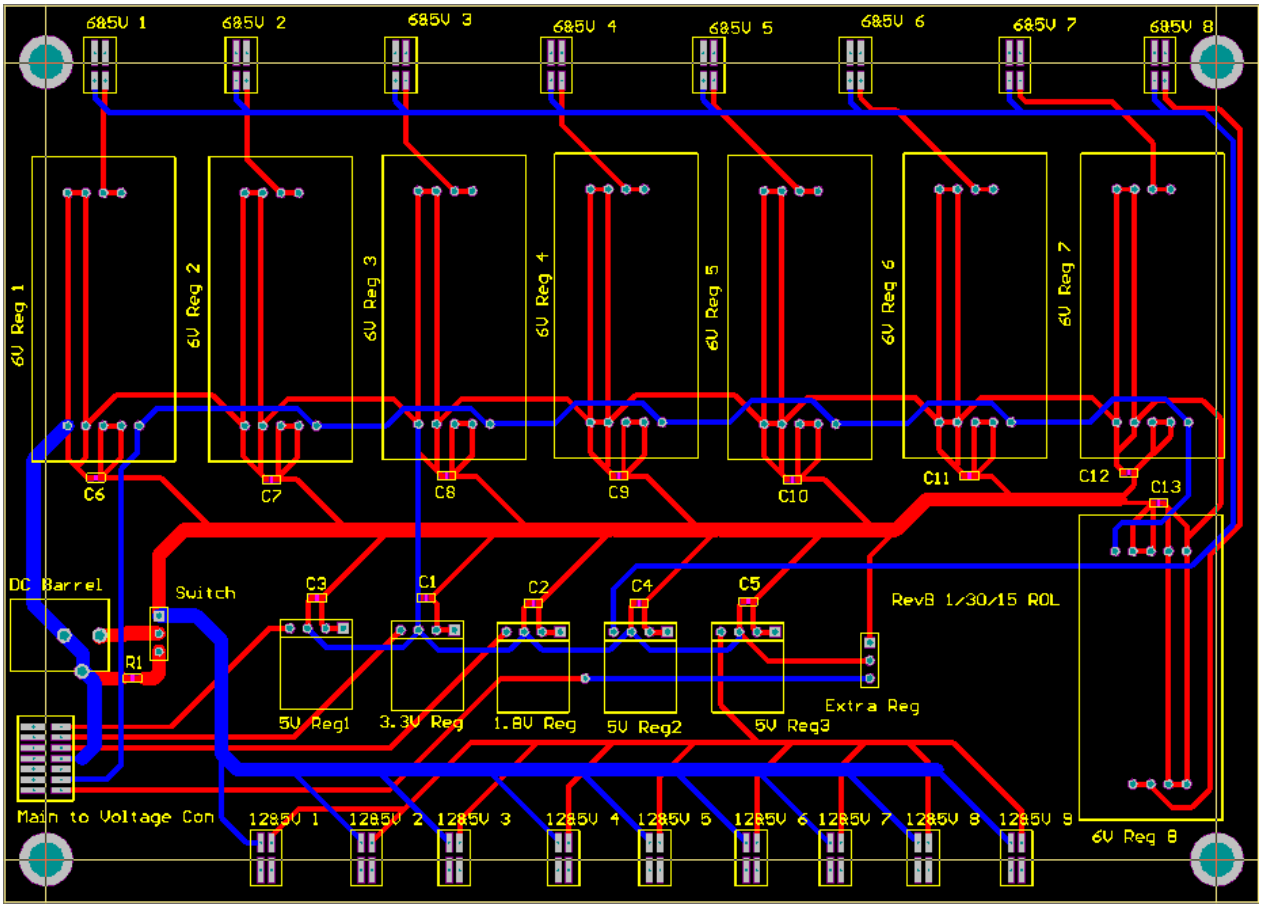



**Figure 43: Voltage Board Revision B**

For the main board, it was found that the footprints for the PWM generators were too small, so there were increased in size. In addition, a bidirectional 3.3V to 5V voltage converter was needed for the communication busses for the SPI to the Raspberry Pi. Also, some of the silk screen resistor numbers were incorrect, so those were fixed. The design for the main board in Revision B can be seen below in Figure 44.

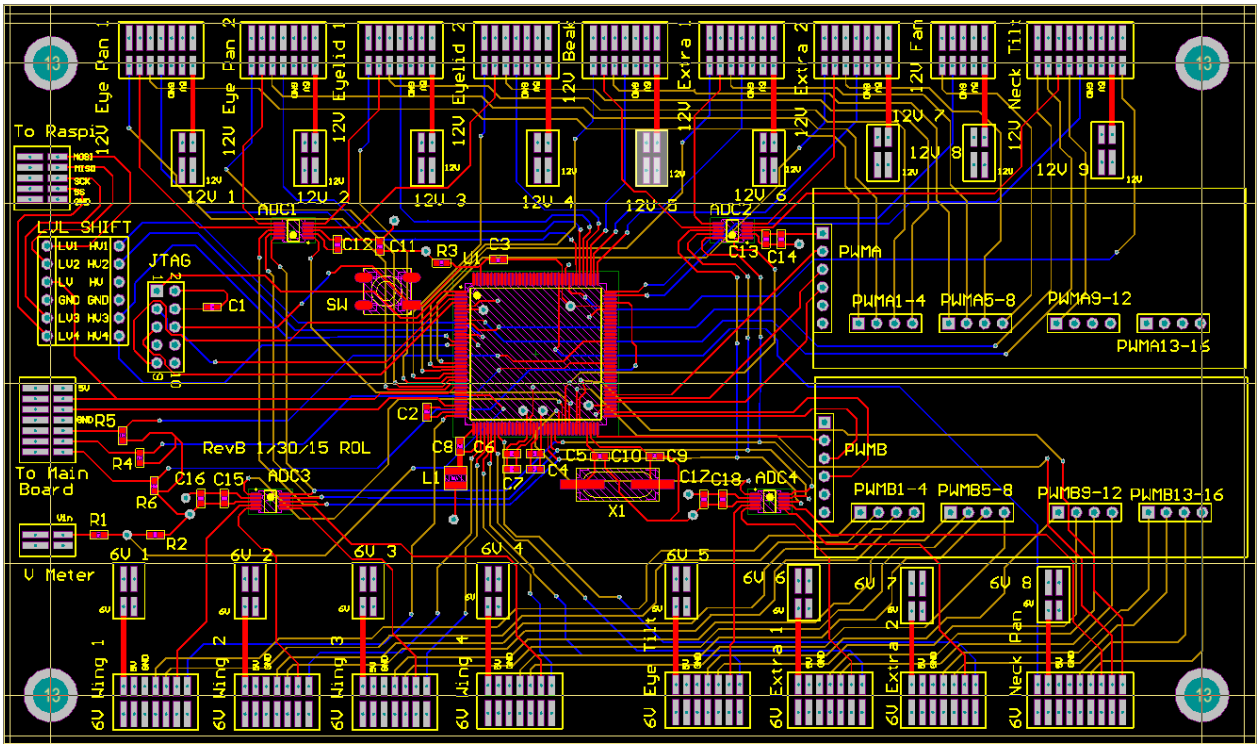



**Figure 44: Main Board Revision B**

Revision B for the motor boards included fixing the pin holes being too small, as with the voltage board, as well as adjusting the mounting hole positions as required by the mechanical constraints, and adjusting the incorrect footprint for the 6V motor driver. Silk screen lines were also added to the sheet to be printed to make for easier cutting of PCBs after printing. The sheet of motor boards can be seen below in Figure 45.

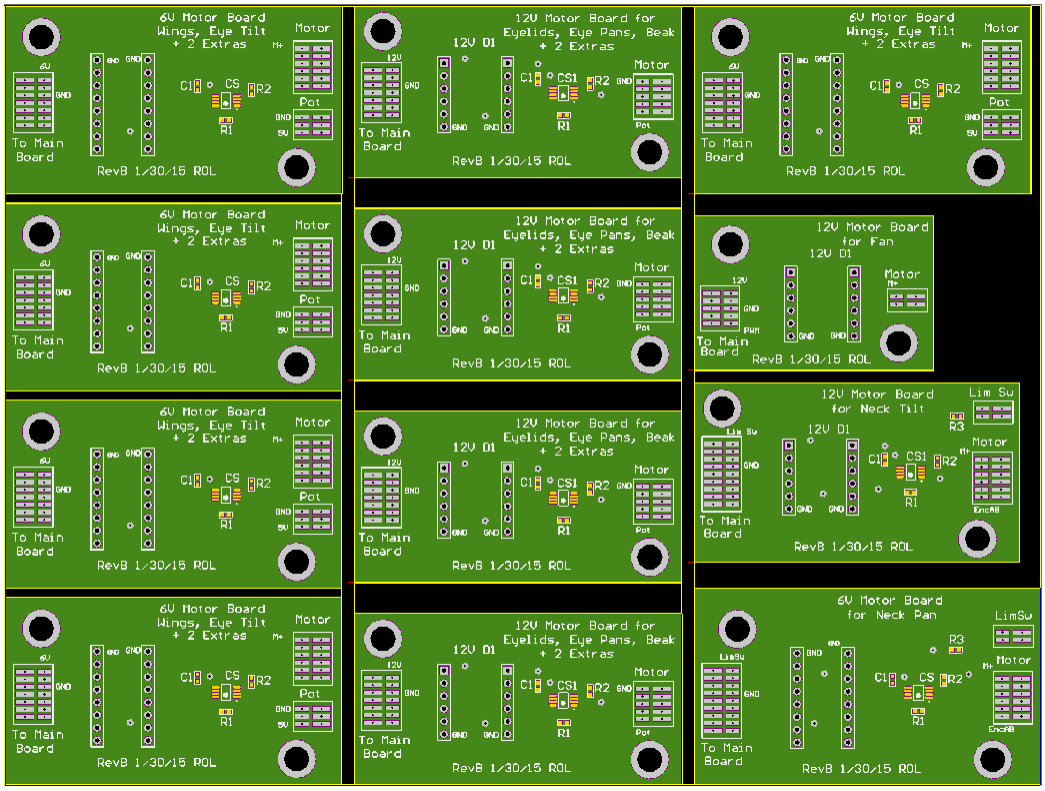



**Figure 45: Motor Boards Revision B**

Revision B is the revision that is currently assembled and operating inside of PABI. The fully assembled main board, voltage board, and five types of motor boards can be seen below in Figure 46 through Figure 52.

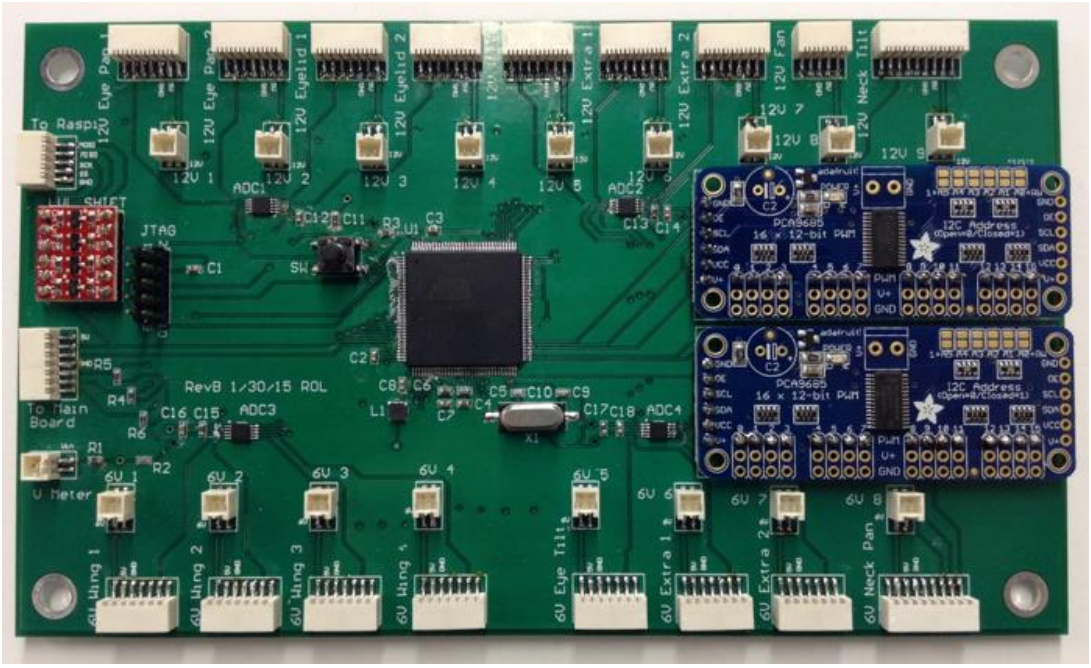



**Figure 46: Assembled Main Board Revision B**

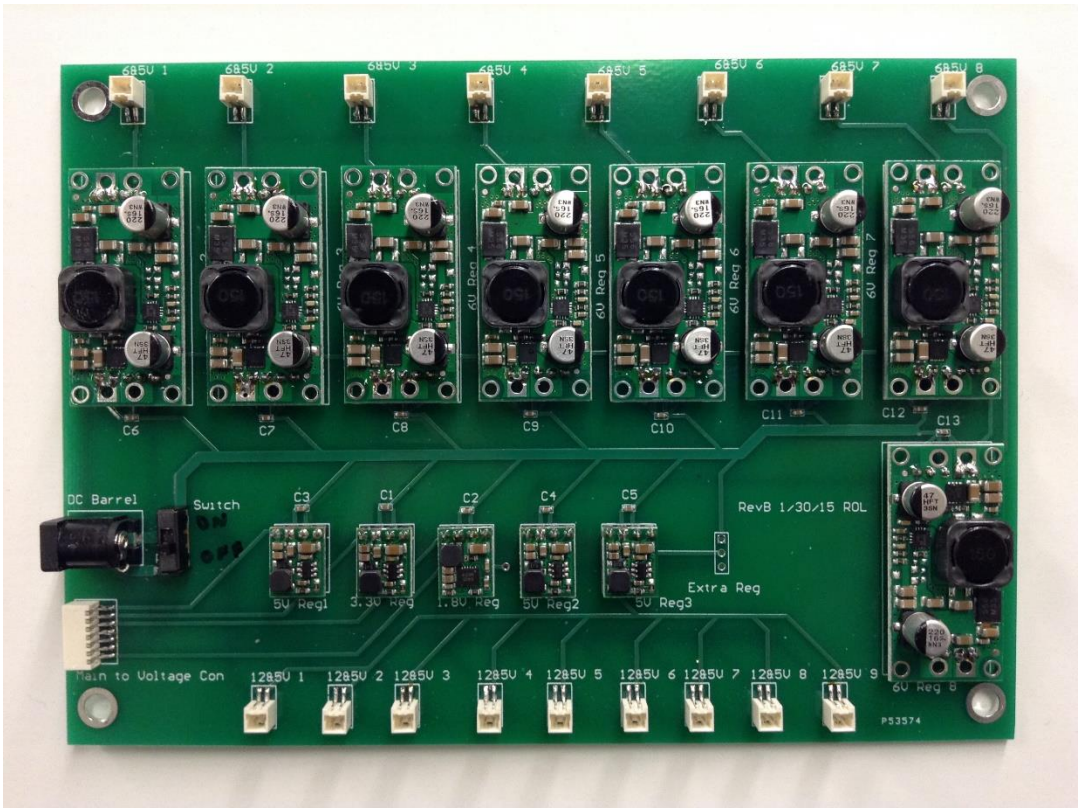



**Figure 47: Assembled Voltage Board Revision B**

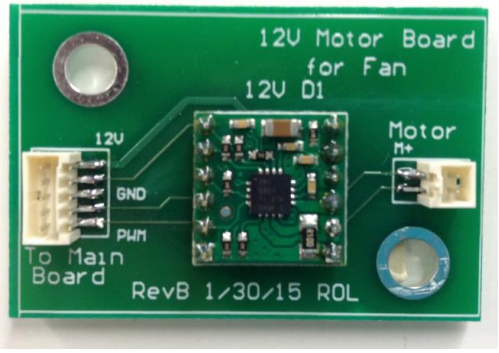



**Figure 48: Assembled Fan Board Revision B**

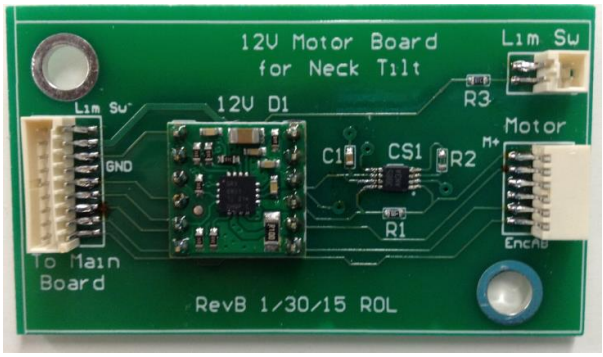



**Figure 50: Assembled Neck Tilt Board Revision B**

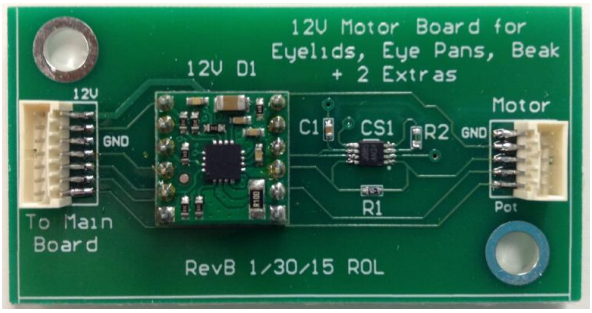



**Figure 49: Assembled Eyelids, Eye Pans, Beak Board Revision B**

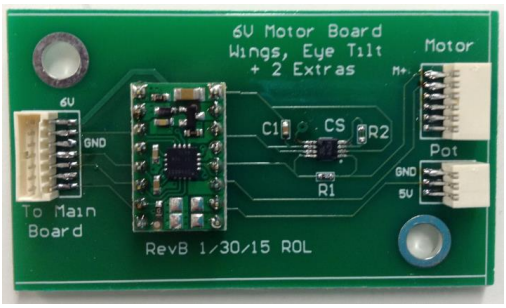



**Figure 51: Assembled Wings, Eye Tilt Board Revision B**

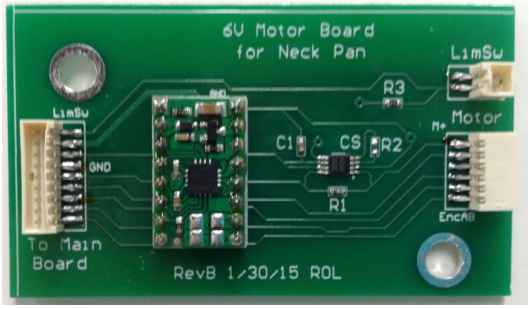



**Figure 52: Assembled Neck Pan Board Revision B**

#### 3.3.4.3 Future Revisions

Revision C has not yet been designed or completed, but a few changes were needed to Revision B the main board and on the motor boards. The biggest change is that a different microprocessor will need to be wired onto the main board, so many of the traces will need to be redone in order to connect it correctly. Next, the I2C data and clock lines on the PWM generators need a 1.5kΩ pull up resistor in order for the I2C protocol to work correctly. In addition, the two PWM generators should be combined onto the same I2C port so that they can use the same bus instead of having two I2C protocols running at once. In addition, the hardware selectable slave addresses on the PWM generators needs to be set to different addresses (0x40) and (0x41). On the motor boards, any board that uses a potentiometer needs to have a voltage divider that brings the 5V logic supply down to 3.3V for the input to the potentiometer so that the reference matches that used in the internal ADC reference. Currently these changes are all implemented using external wires and resistors attached to the Revision B boards, but in future work new boards should be printed for neatness.

### 3.3.5 Embedded Design

#### 3.3.5.1 SPI Communication between the Microcontroller and the Sub-Computer

The SPI communication between the microcontroller and the Raspberry Pi is the most vital communication protocol, since it enables the main computer to communicate with the microprocessor. This communication protocol had to be able to transmit motor command positions as a value from 0 to 255 from the Raspberry Pi to the microprocessor as well as current motor positions and motor statuses from the microcontroller back to the Raspberry Pi for use by the main computer.

##### 3.3.5.1.1 Protocol

In order to successfully transmit the necessary information between the sub-computer and the microprocessor, the following SPI protocol was established. First, the sub-computer acts as the SPI master, providing the clock signal and choosing when transfers occur. The microprocessor acts as the slave with an interrupt tied to its slave select pin so that it can read transfers immediately once a transfer starts. After pulling the slave select low, the sub-computer waits for two clock cycles in order to allow for the interrupt service routine on the microprocessor to start, and then transmits a total of 50 bytes. The first 2 bytes are dummy bytes for timing purposes, and the next 24 bytes consist of twelve motor IDs and positions, where every first byte is a motor ID and every second byte is the corresponding

desired motor position. This position is a value from 0 to 255, which is later converted to potentiometer values in other parts of the code. In the future, this packet should be expanded to contain more information, such as the status of motors and errors from the motors such as if a motor is stalling. During the transmission of these first 24 bytes, the microprocessor simply receives the data. The following 24 bytes sent by the Raspberry Pi are dummy data, and the microprocessor sends motor ID and current motor positions. As with the previous data, every first byte is a motor ID and every second byte is the corresponding current motor position. Figure 53 provides a visual aid for understanding this protocol.

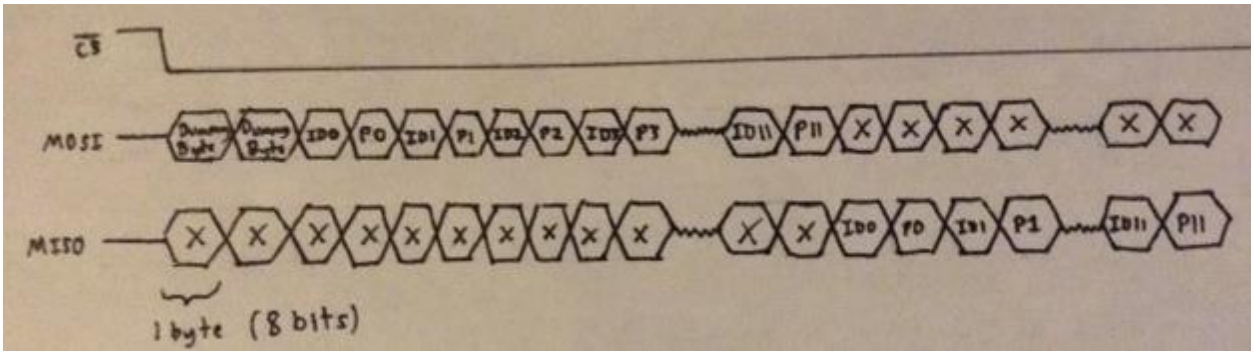



**Figure 53: SPI between sub-computer and microprocessor**

#### 3.3.5.1.2 Timing Information

In order to successfully transmit data, the clock frequency must be usable by both sub-computer and the microprocessor. The sub-computer is capable of outputting a number of SPI clock frequencies by using only prescalers. As the slave, the microprocessor is capable of receiving any clock input below its operating frequency of 55MHz, though lower frequencies are received more reliably. The clock frequency decided on was 488 kHz based on the timing requirements of the system. Slower data transmission resulted in less chance of data corruption, but a faster speed could have been chosen in order to speed up the system. Other methods could be used to avoid data corruption, such as adding in a test byte such as 01010101, but in this implementation there is no error checking, and so transmitting at slightly slower speeds gives the highest probability of accurate packet transmission.

#### 3.3.5.1.3 Sub-Computer Code Implementation

In order to implement the above protocol on the sub-computer, a Raspberry Pi model B+, the first thing that needed to be done was the installation of Raspbian as the operating system. The next step was enabling the use of the SPI port, which is done by following the following steps:

1. Open Terminal on Raspberry Pi
2. Type Command: sudo raspi-config
3. Navigate to Advanced Options
4. Navigate to SPI
5. Enable SPI
6. Finish

Once SPI was enabled on the Raspberry Pi, the BCM2835 Library was downloaded in order to provide access to the hardware registers. The structure for sending and receiving the 50 bytes described in the protocol section (2 dummy bytes + 24 motor IDs and positions from Raspberry Pi + 24 motor IDs and positions from the microprocessor) is as follows. First, functions from the BCM2835 library were used to initialize the SPI in master mode, set the clock frequency, have data send on the rising edge of the clock signal, and have two clock cycles between the slave select dropping and the first transfer. The code then initiates sixty 8 bit transfers, the first 24 of which contain information to send the microprocessor, and the last 36 of which are dummy data. With each of the last 36 bytes, the Raspberry Pi stores the bytes to be used later by the higher level functions.

#### 3.3.5.1.4 Microprocessor Code Implementation

On the microprocessor, there were a couple of steps to set up the SPI communication for slave mode. The SPI busses that are connected to the Raspberry Pi are using SPI0, so the initializations start with initializing the GPIO pins as well as the SPI0 module. The GPIO pins for MISO, MOSI, SCK, and SS are set as an output, an input, an input, and an input with interrupt capabilities respectively. In order to initialize the SPI0 module properly for this slave mode, the following bits are assigned:

| | |
|---|---|
| Mode Register: | MSTR bit = 0 (Initialize in slave mode.) |
| Chip Select Register 0: | BITS bits = 0x0000 (Bits per transfer is equal to 8.) |
| Interrupt Enable Register: | NSSR bit = 1 (Enable slave select interrupt.) |
| Control Register: | SPIEN = 1 (Enable the SPI after initialization.) |

In order to receive and send data when initiated by the sub-computer, an interrupt on the slave select pin is used. This interrupt is initialized as the highest priority interrupt so that it will minimize latency as much as possible, allowing for more frequent transmissions. It also is initialized specifying the SPI0 interrupt vector of 736. This interrupt activates whenever the slave select pin is pulled low and then immediately starts checking the RX register to see if a byte has been received. For the first 24 bytes, each time a byte is received it is stored in an array that is later used to assign motor positions. For the last 36 bytes, it does the still waits for bytes to be received except that it sends its own data across and doesn't store what the Raspberry Pi is transmitting.

### 3.3.5.2 SPI Communication between the Microprocessor and the External ADCs for Current Sensing

The SPI communication between the microprocessor and the external ADCs is the lowest priority task to accomplish in this design, as they are connected to the current sensors which functionality are being added for use in the future. However, the SPI protocol and timing for communication with the ADCs can be seen below. This communication protocol has been tested and works as intended.

#### 3.3.5.2.1 Protocol

The ADC124S051 is a 12 bit resolution analog to digital converter which communicate using SPI. After the slave select pin is pulled low, two bytes can be sent to the ADC. The first byte sent must have

bits 3, 4, and 5 used to designate the channel to receive from. Both bytes should be read back by the microprocessor, as bits 0, 1, 2, and 3 contain the most significant bits of the ADC value, and the second byte contains the least significant bits. In this way, the 12 bit number corresponding to the ADC value can be recorded. This protocol can be seen below in Figure 54.

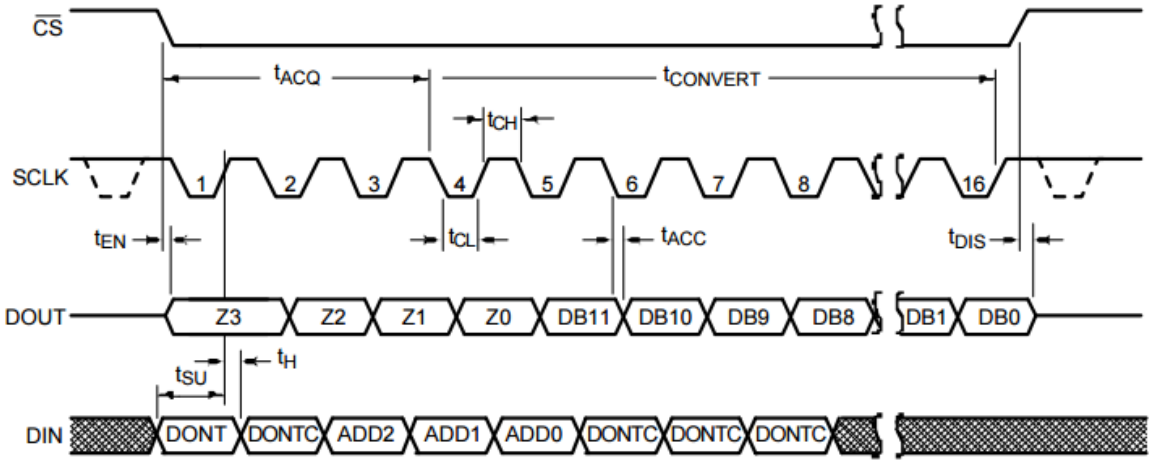



**Figure 54: ADC124S051 Protocol (ADC124S051 Datasheet)**

#### 3.3.5.2.2 Timing

The maximum speed that the ADC124S051 can operate at is 8MHz, so the maximum clock signal that can be used to send SPI information at is an 8MHz clock frequency. Ultimately a different frequency could be chosen after taking the calculations in the Overall Timing section into consideration.

### 3.3.5.3 I2C Communication between Microprocessor and PWM Generators

The I2C communication between the microprocessor and the PWM generators what enables the movement of the motors. There are 13 motors, seven of which require one PWM signal, and six of which require two PWM signals. In order to provide all of these PWM signals as fast as possible whenever they are needed, two 16 channel PWM generators are used. Both communicate with the main processor via I2C (or TWI as it is known on the main microcontroller).

#### 3.3.5.3.1 Protocol

When using two I2C slave devices, they are both typically connected on the same I2C bus, and each have different slave addresses. I2C devices have two wires, the data line and the clock line. The data line is bidirectional, and the clock line has a clock signal that is supplied by the master. The way that I2C works is it first transmits a start bit, followed by a 7 bit slave address, followed by a read/write bit. Once this address is sent, only the I2c slave with that slave address will read the data sent by the master, and that slave responds with an acknowledgement by pulling the data line low for one clock cycle. In this way, two devices connected to the same I2C bus can be written to independently of each other. The master then transmits as many bytes as it would like, each followed by an acknowledgement, until the master is done transmitting, at which point it sends a stop bit. In order to differentiate between the two PWM generators, the slave address can be changed using hardware on the Adafruit PWM generator breakout board for the PCA9685.

The PCA9685 has 16 output channels. Each is governed by two 12 bit inputs. The first input specifies how long the signal should be high and is a value ranging from 0 to 4095, where 4095 is the number of clock cycles for each PWM pattern. The second input and the other saying how long the signal should be low and is a value equal to 4095 minus the high time value. Each of these 12 bit values is split into two registers, one for the four most significant bits and one for the eight least significant bits, meaning that each PWM channel has four registers that must be written to in order to achieve a desired PWM signal. There is also a maximum frequency register, as well as two mode registers which govern the method of operation of the PWM generator. Most notable for our application is the auto-increment feature, which allows for writing to multiple sequential registers without having to specify each individual control register by automatically incrementing the control register after each byte is written to. In order to write to a register, the master must transmit the start bit, slave address, read/write bit, the control register of one of the registers, and then the value to write to that register. This can be seen below in Figure 55.

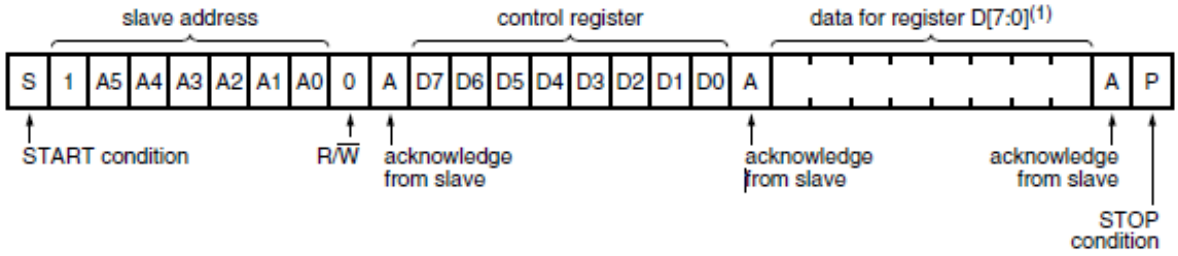



**Figure 55: Write to a Specific Register on PCA9685 (PCA9685 Datasheet)**

When writing to the four registers that govern a PWM channel, the sequential order of the registers is high least significant bits, high most significant bits, low least significant bits, and low most significant bits. When using the auto-increment feature, the master can write the control register of the first of these, followed by the four bytes corresponding to those four registers, followed by the stop bit. A diagram showing how to write to all PWM (called LED in the datasheet) registers sequentially using auto-increment is shown below in Figure 56.

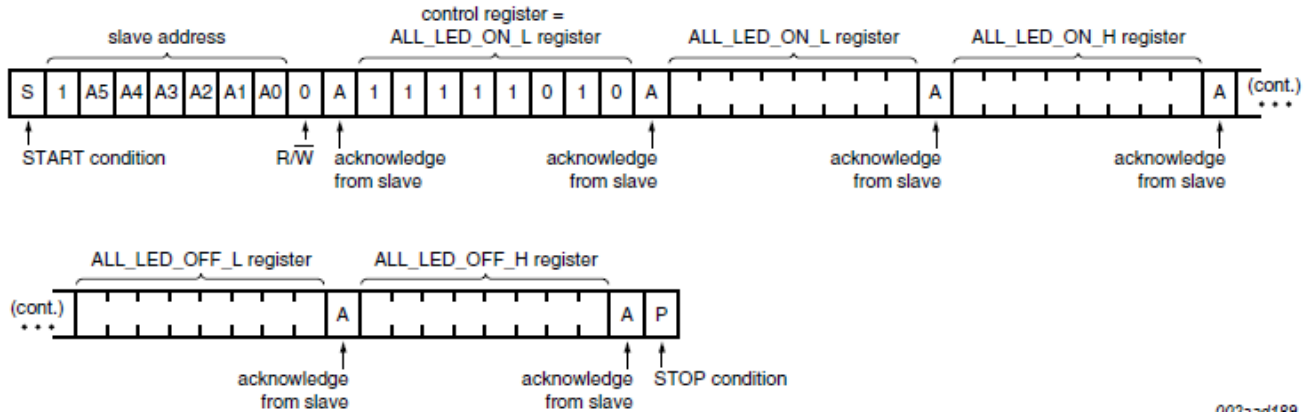



**Figure 56: Writing to all PWM (LED) Registers Sequentially on PCA9685 (PCA9685 Datasheet)**

#### 3.3.5.3.2 Timing

The timing for the PCA9685 is such that the maximum frequency that the clock can be provided at is 25MHz. Other lower frequencies can be used as well, and the frequency chosen for communication with the PWM generators was 8MHz. This was due to lax timing constraints as discussed in the Overall Timing section, and a slower communication speed results in more reliable communication.

#### 3.3.5.3.3 Microprocessor Code Implementation

The implementation of the PWM generator protocol for writing to one of the PWM channels contains a couple of key parts. First, the GPIO ports with the I2C data and clock lines on them needed to be initialized and enabled. The data line is initialized as being bi-directional, while the clock line is initialized as an output. Next, in order to set up the I2C for the desired speed of 8MHz, the built in I2C setup function (referred to as TWI on microprocessor) is called with the proper TWI instance (TWI module 1) and the speed to initialize it at. In this function, the internal clock for the TWI1 module is activated, the interrupt for receiving acknowledgements is activated, and the basic mode bits are set to put the module into master mode with the desired speed. The ISR is a prebuilt function that checks for acknowledgments from the slave when necessary. In order to write to the data bus, a built in function for writing is used. This function takes in the slave address of the device to send to, the control register on the slave device to start at, and an array of bytes to send over the data bus. The write function handles the start, stop, and read/write bit. In order to make writing to a specific control register of either PWM generation device simpler, a function called writeToPWMReg was created. This function takes in the slave address of the PWM register to write to, the PWM generator control register to write to, and the data to write to that register. It then uses the built in write function to write to that register.

In order to initialize the PWM generators in the format most useful to this application, the MODE1 register was written to with all of the normal mode functions. Then, the PRESCALE register was written to with the value calculated in order to have the PWM frequency be set to 25MHz. Finally, the auto-increment bit in the MODE1 register was set so that in the future, one control register could be specified, and it would automatically increment the control register after each byte that was sent. In this way, only one write function would be needed in order to write to all four registers necessary to assign a duty cycle to one of the PWM channels. These initializations were performed on both PWM generators.

Finally, since the purpose of these PWM generators is to output a duty cycle specified by the user for controlling the motors, a function for setting the duty cycle to a specific PWM channel was created. This function takes in a PWM generator slave address, the PWM channel number to be written to, and the duty cycle as a value from 0 to 100. The function then sets the high time to the max of 4095 multiplied by the duty cycle, and sets the low time to the opposite. In order to write to the four required registers, the least significant bits of the high time are sent first, followed by the most significant bits of the low time, followed by the least significant bits of the low time, followed by the most significant bits of the high time. This is done using the auto-increment feature so that only the first control register must be specified.

### 3.3.5.4 Internal ADCs

The internal ADCs on the main processor are responsible for reading the potentiometer values for every potentiometer that is connected to a motor. These ADCs can operate in either 8, 10, or 12 bit mode, but 12 bit mode was chosen for its higher precision. According to the AT32UC3C0512C datasheet, the time that it takes for one 12 bit ADC channel to be read is 5.3us in a worst case scenario. We attempted to program the internal ADCs to receive values from any ADC channel, but an unexpected error prevented the ADCs from working during this iteration of PABI. Specifically, when using the sample code provided by Atmel, only the ADC channels that are preloaded into the sample code (5 and 14) can be read. If another ADC channel is swapped in for one of the defaults, the value floats around ground. We believe this to be due to an internal reference somewhere not accessible in the main portion of the sample code. In addition, when the exact code from the sample was transferred into the main PABI project, ADC channels 5 and 14 exhibited the same behavior as the other ADCs, and had values floating around ground. Once this was found to not work, support for initializing the ADCs at the register level was searched for, but no support was found for using the AT32UC3C's ADC module for channels other than 5 and 14 was found, and the support forums indicated it to be a common issue. As a result, we recommend using a different processor with a more reliable ADC module. Alternatively, if a faster way of communicating with Atmel support could be found the issue may be able to be solved, but the average response time of Atmel support was 4-5 days, which for the timeline of this project was not useful for a prolonged debugging session.

### 3.3.5.5 Internal Quadrature Decoders

The internal quadrature decoder channels on the main processor are responsible for reading the values of the two quadrature encoders that are used on the neck pan and neck tilt motors. These position values have a 16 bit resolution according to the AT32UC3C0512C datasheet. The code for the decoders will be added in future iterations of PABI.

### 3.3.5.6 Overall Timing

In order to determine what speeds to have the different communication protocols operate at, it was necessary to determine the minimum and maximum amount of time each one could take. We also had to estimate the approximate times for other functionalities so that a range of usable transmission speeds could be determined. In this way, slower and more reliable speeds can be used when possible, while still staying inside the timing requirements of the system.

#### 3.3.5.6.1 Timing Calculations

##### 3.3.5.6.1.1 SPI between Sub-Computer and Microprocessor

The sub-computer could operate at up to 125MHz and the microprocessor could operate at up to 55MHz. The library used for SPI communication with the sub-computer supports prescalers for the clock frequencies shown below in Table 9.

| SPI Clock Frequency | SPI Clock Frequency |
|---|---|
| 125MHz | 976kHz |
| 62.5MHz | 488kHz |
| 31.2MHz | 244kHz |
| 15.6MHz | 122kHz |
| 7.8MHz | 61kHz |
| 3.9MHz | 30.5kHz |
| 1953kHz | 15.2kHz |

**Table 9: Raspberry Pi SPI Clock Frequencies**

Any of these frequencies that are below 66MHz are readable by the microprocessor, which has an internal clock that oscillates at 66MHz, so the fastest usable clock frequency from the sub-computer is 62.5MHz. This is very close to the maximum internal frequency of the microprocessor, however, so we calculated at the next highest useable frequency. At this frequency, a bit is transferred every 32ns. There are 8 bits in a byte, and the previously discussed protocol for this SPI bus consisted of 60 bytes being sent. In addition, it takes 1 clock cycle for the slave select to be pulled low, and the Raspberry Pi then waits for 2 clock cycles to send its information. Therefore, the number of clock cycles it takes to send and receive all sixty bytes is equal to: (8 * 60) + 1 + 2 = 483 clock cycles. At a frequency of 31.2MHz, this will take 15.46µs. This is the fastest that the communication between the Raspberry Pi and the microprocessor can occur, but using a lower frequency would be preferable due to the decreased risk of dropped or erroneous packets. Table 10 below shows the different amounts of time the communication process would take with different SPI clock frequencies.

| SPI Clock Frequency | Time for SPI Communication |
|---|---|
| 31.2MHz | 15.46us |
| 15.6MHz | 30.96us |
| 7.8MHz | 63.55us |
| 3.9MHz | 123.85us |
| 1953kHz | 247.31us |
| 976kHz | 494.88us |
| 488kHz | 989.75us |
| 244kHz | 1.98ms |
| 122kHz | 3.96ms |
| 61kHz | 7.92ms |
| 30.5kHz | 15.84ms |
| 15.2kHz | 31.78ms |

**Table 10: Raspberry Pi to Microprocessor Communication Times**

#### 3.3.5.6.1.2 I2C Communication between Microprocessor and PWM Generators

The I2C communication between the microprocessor and the PWM generators can have a maximum clock frequency of 25MHz. In order to communicate with any given channel, the microprocessor must send out 6 bytes plus the start and stop bits, as well as wait one clock cycle for an acknowledgement after the first 5 of these bytes. This adds up to (8*6) + 1 + 1 + (1*5) = 55 bits, which means that the transfer to write to one PWM channel takes 55 clock cycles. At 25MHz, this takes 2.2µs, but as with the SPI communication, a slightly slower frequency is preferred to avoid the possibility of

dropped or erroneous packets. The microprocessor can choose almost any clock frequency below 25MHz, so the possibilities are fairly open for the I2C communication clock frequency.

#### 3.3.5.6.1.3 SPI Communication between External ADCs and Microprocessor

The external ADCs that communicate via SPI with the microprocessor can read an SPI clock frequency of 3.2MHz to 8MHz. Reading one ADC channel requires 2 bytes. In addition, one clock cycle is required for pulling the slave select low, and one clock cycle must follow before the two bytes are transmitted. This means that 18 clock cycles are necessary for reading each ADC channel, which at 8MHz takes 2.25µs.

#### 3.3.5.6.1.4 Internal ADCs, Quadrature Decoders, and Other Considerations

According to the datasheet for the AT32UC3C0512C, the time it takes to read each internal 12 bit ADC channel is 5.3µs. There is no set time for reading the quadrature decoders, but 6us is a good conservative estimate. Time must also be added for miscellaneous calculations and data movement, which are accounted for in the next section.

### 3.3.5.6.2 Timing Conclusions

In order to determine approximately how long the program will take to run, a number of different items must be accounted for. In order to get smooth motion on all motors, we decided that it would be best to have all velocities for all motors inside of this PID loops must be able to update at least 10 times between each SPI update from the sub-computer. In addition, each current sensor must be checked at least once (external ADC channels), as well as the two limit switches. There are seven PID loops that use 6V motors which require two PWM signals and potentiometers to measure position, and seven that use 12V motors which require one PWM signal and potentiometers to measure position. There is one 6V motor that requires two PWM signals and an encoder reading, and one 12V motor that requires one PWM signal and an encoder reading. Table 11 below shows the timing for each of the PID loops.

| Motor Type | Quantity | PWMs (2.2us each) | Potentiometers (5.3us each) | Encoders (6us each) | Total Time Per Motor | Total Time For All Motors |
|---|---|---|---|---|---|---|
| 6V, Potentiometer | 7 | 2 | 1 | 0 | 9.7us | 67.9us |
| 6V, Encoder | 1 | 2 | 0 | 1 | 10.4us | 10.4us |
| 12V, Potentiometer | 7 | 1 | 1 | 0 | 7.5us | 52.5us |
| 12V, Encoder | 1 | 1 | 0 | 1 | 8.2us | 8.2us |

**Table 11: PID Loop Timing**

All together, the PID loops take 139µs. In order to have enough time allotted for the PID calculations in each loop, 5µs per PID loop is allotted for mathematic operations. This brings the time per set of PID loops up to 219µs. In order to allot time for 10 iterations of the PID loops, this is multiplied by 10, bringing the total up to 2.19ms. The positions are updated at approximately every $10^{th}$ loop. There are 16 current sensors to read, which adds on 36us, as well as another 50us added on for any other writing or math operations that might need to be done. This brings the code time up to 2.28ms. Adding on the 2.25µs it takes for a single full update between the sub-computer and the microprocessor as discussed in the timing for the SPI protocol, the total time is brought up to 2.29ms for each iteration

of the code. This is just approximate, however, as once the PID loops are implemented in future iterations of PABI the timer for the PID loops will be configured to be approximately the time it takes for all of the motors to update (219us), and will likely be a few micro seconds longer.

The high level requirement is that new position data be sent from the Raspberry Pi to the microprocessor at a minimum of 30Hz in order to match the video frame rate for eye tracking, meaning that the SPI transfer must start every 33.3ms, even though the current i7 main computer implementation only transmits as it is currently configured at a frequency of 4Hz. With the estimate for the fastest execution of the microprocessor being 2.29ms, there is a little bit of room to slow down some of the systems without a detriment to performance. The most vital protocol to slow down is the SPI between the sub-computer and the microprocessor. At 488 kHz, the execution time of the microprocessor code rises to 3.27ms, which still leaves a very large margin for error. It also makes the SPI communication much more reliable and less likely to have dropped or erroneous packets. Another communication protocol that can be slowed down is the I2C, which if it is slowed to a frequency of 8MHz (and using the new SPI speed), only brings the entire execution time up to 4.39ms. The rest of the systems are already at a good speed, so these timings work. With these frequencies, the 30Hz requirement can be increased up to 200Hz and still meet all of the timing requirements.

## 3.3.6 Control System Design

### 3.3.6.1 Motor Control

This section discusses the higher level design involved in the motor control. The overall flow is that the desired motor positions are received from the Raspberry Pi, the set points in the PID are updated, and the PID control loops set the corresponding PWM duty cycles to make the motors go towards their set points. The positions received from the sub-computer are on a scale from 0 to 255, where 128 is the neutral 0 position of the motor, and the all values are mapped to the possible motion of each motor in potentiometer ticks. This was not implemented in this iteration of PABI, however, and must be done in order to implement closed loop control.

#### 3.3.6.1.1 Motor Struct

In order to make coding simpler and more versatile, a motor struct was created in order to predefine all of the motors being used and their corresponding PWM channels and whether they were 12V or 6V. It also contained their current sense channel information and their potentiometer or encoder information. In this way, each motor could be called by name and not have to worry about mixing up any data channels. It also contains all of the information needed for PID control. The structure can be seen below in Figure 57.

```
typedef struct {
        int sixOrTwelve;
        uint8_t pwm1;
        uint8_t pwm2;

        uint8_t currentSense;

        int hasPot;
        uint32_t potPin;
        int hasEncoder;
        uint32_t encAPin;
        uint32_t encBPin;

        uint8_t desiredPos;
        uint8_t currentPos;

        uint32_t pGain;
        uint32_t iGain;
        uint32_t dGain;
```

**Figure 57: Motor Struct**

#### 3.3.6.1.2 Setting Duty Cycle

In order to set the duty cycle of any specific motor, a function was made that takes in a motor, the duty cycle it should be operating at, and the direction that the motor should turn in. This function then checks the motor to see if it is a 6V or a 12V motor, and then chooses the way it will assign the duty cycle based off of that. If the motor is a 6V motor, the duty cycle and direction is decided by setting one of the PWM channels connected to it to 0V and the other to output the desired duty cycle. If the motor is a 12V motor, it assigns the duty cycle to its one PWM channel, and toggles the digital logic direction pin to have the motor change directions.

## 3.4 Software Design

### 3.4.1 The Software Architecture

Restructuring the software architecture of PABI was a major aspect of this project. We redesigned the architecture because the devices on PABI's older revisions were not only used old versions of software, but also used old, outdated devices.

The previous system running on PABI used Robot Operating System (ROS) and an android tablet. PABI initially was running ROS version Electric on Ubuntu 11.10, which is very out of date. The system relied heavily on ROS packages and used the ROS package protocol to transmit data between subsystems. Most of the ROS programming was done in Python 2.7, and the interfacing application on the tablet was programmed in Java. Beyond the ROS layers in the embedded systems, C and AVR were likely used.

Despite overhauling and redoing most of the code on PABI, some general ideas were taken from the older versions of PABI. The idea of a subscriber/publisher architecture is ideal for inter-computer/inter-process/inter-system communication. The older system also had the idea of distributing low level processing through intermediary devices for specific actuation. The main drawback of this design was that we could not find accurate documentation on the exact specifics of this architecture and its implementation. What could be discerned was found through code analysis.

A lot of the concepts were brought back in the new system architecture but made expandable for the future. The new system kept the subscriber/publisher network style of inter-system communication for easy debugging, and standardization of communication. The latter is a powerful idea because everything can now communicate through a network and not through a direct connection, such as a USB device. This subscriber/publisher idea also allows for easy debugging, as it is easy to "listen in" on a topic and receive what is passing through the system. The team developed a distributed system that communicated over a network centralized on this method of communication. Figure 58 shows the proposed architecture.

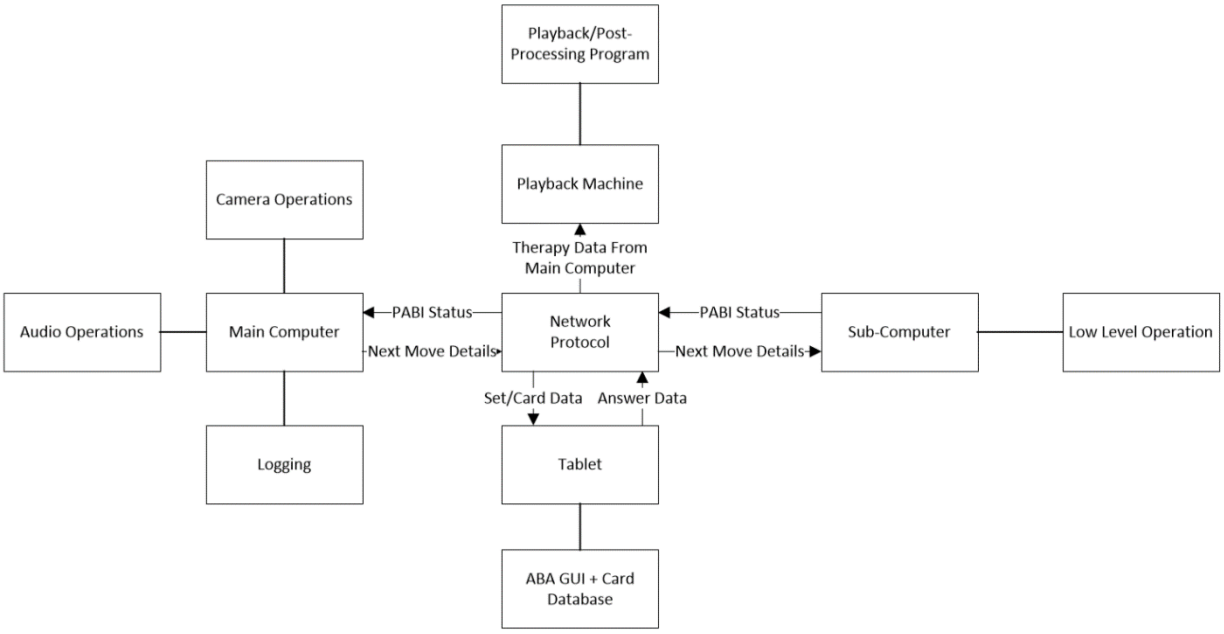



**Figure 58: Overall distributed structure and communication messages between members**

Some benefits in this design included that none of these items needed to actually be placed inside of the robot. Since the system communicated via network protocols, the main computer could be in the cloud, or elsewhere as well as all other peripherals. The only device on this list that truly needs to be inside of the robot is the sub-computer and the lower level items that actuate the robot. This opens the door to expansion for later projects and systems. Another important point that this design is the ability to statically update individual parts of the robot and have it not affect other code in the pipeline. For example, if finer control were to be added, the editor would only need to edit the sub-computer code without fear of disrupting the main computer or tablet code. Lastly, this design allows any device to be the master controller with almost no changes to the command structure.

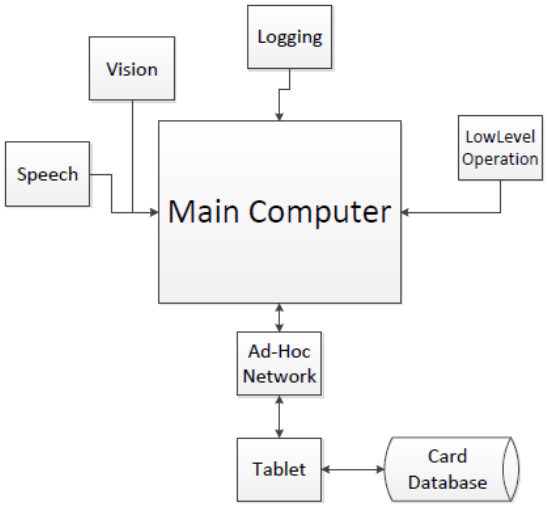



**Figure 59: Final, monolithic design of PABI**

The biggest drawback to this design is updating the physical networking structure messages in the system. A small change in one message requires that the marshaling change on both ends of the pipe to accept the change, leading to headaches when expanding. A monolithic design was considered because it removes the penalty of code conflict on expansion. A monolithic design also removes the connection between the main computer and the sub-computer through combining the actions of both into one master computer. The advantages of this design included removing network links between subsystems, and having less transmission latency between commands and control. One drawback is the difficulty in updating the software on these combined parts. With this all-in-one system, there were many dependencies on control and implementation sprinkled throughout threads. An innocent change in parameters or implementation could require system wide changes to actually implement using this architecture.

Weighing the consequences of both systems, the team selected the distributed design described above in Figure 59. While there are some penalties in updating the network pipelines, it seemed the most manageable and sustainable for future projects to expand on. For example, this design allows for easy portability of devices, or even removal of devices with the cloud.

### 3.4.2 The Protocol

The network structure was critical to ensuring that this framework will be used for years to come. The team presented three very different designs, detailed below.

The first solution the team considered was reusing the ROS protocols from the previous project. This would have been the least amount of work, and was guaranteed to work over wireless and wired networks. This protocol also came prebuilt with a lot of commands and packages to use and make other parts of the project easier if the correct hardware was used. A major downside of recycling would be the cost of updating the older versions of ROS to a more recent revision. Another downside was that ROS is an active project and can sometimes become unstable and unsupported by operating systems. This is

not sustainable for future projects as it would not only slow our group down, but it will also slow down the following groups. The ROS packages that come with ROS are not only voluminous, but also add a lot of bloat to the install if we are just using it for network protocol. Lastly, ROS is not effective on any other operating systems other than Ubuntu. This will limit the drivers and the machines that can interface with PABI in the network without having extra applications that translate the data to and from ROS.

A second option that was considered was custom making the network protocol through socket programming. This would ensure that we have control over the source and that the only space it would take up on the computers involved would be transmission protocol code. This method also allowed any operating system to communicate with any device on the network with its own special protocol. This method, however, will take up an enormous amount of time to develop and would have had some bugs at the end. Another issue is creating the topic/subscriber model described earlier in this section. Simply put, this method was not feasible in the 21 weeks of the project, and did not hold enough integrity to be used for future project expansions.

The last option observed was utilizing a library called Lightweight Communications Marshaling (LCM). This is a tried library used by many different institutions around the world for exactly as its name implies (Developers, LCM 2015)**.** This library uses a very similar protocol to ROS without the extra packages. This library is also very compatible with many programming languages, and is portable to most popular operating systems with incredible documentation. Lastly, this library takes care of all transmission structures and UDP marshalling for the programmer with only a description of the structure to send. A major flaw to this library is that it is inefficient on some operating systems, Windows in particular, and does not hold a packet drop protocol built in. This deficiency can be an issue over wireless communications.

It was a tough decision between ROS and LCM, but the team selected LCM for use in PABI. While the lack of extra packages and wireless protocol might seem like obvious strong points ROS has over LCM, the stability and pedigree trumps any extra features. The cost of rewriting every subsequent improvement's network code is unacceptable if PABI is to continue into the foreseeable future. The cost of binding everything to Linux is also very high if this product were to be used in the field with ROS, as the number of average people that use Linux is very low compared to those of other more popular operating systems (NetMarketShare, 2015).

### 3.4.3 The Devices

#### 3.4.3.1 Sensing Devices

A large amount of the perceptual data from PABI will be received through the camera sensing devices. These cameras need to not only complete the objective of the project, but also need to allow expansion for future projects on this platform. Since the main RGB cameras were positioned in the eyes, they needed to be small, or have small circuit boards that could fit inside of the eyeballs. These cameras also had to be able to detect faces, so the resolution needed be high enough for accurate detection. For this, we selected a 720p webcam distributed by Optimum Orbis. Stripping away the covers and lights allowed the cameras to fit inside of the eyeballs effectively without loss to performance. The resolution

was also very high (720p) and the lens was manually focused, allowing for stereoscopic depth detection in future projects.

Instead of using the 720p cameras for stereoscopic vision, the team decided to use the Senz3D by Creative to get depth and gesture data. Under the Windows operating system, the Intel Software Development Kit (SDK) allows this device to recognize gestures, as well as RGB with face recognition. We attempted to use this device because it had more potential for expansion than just the two 720p cameras. During the project, Intel dropped support of this device in favor of their new camera to be released in 2015. Given this lack of support, this device was eliminated from the final robot.

#### 3.4.3.2 Computing Devices

The choice of main computer for this project was critical, as a poor choice here could have disrupted or even destroyed the semi-distributed architecture planned. The device needed to be able to handle many intensive threads and also read and write to the disk very quickly. This computer was the nucleus of networking, servicing and producing requests on the network for other devices and peripherals used. Given these conditions, something with large processing power, fast disk, and high network capability was required. A first choice was to stick a laptop into the robot, but due to size constraints, this was not possible. The next best option was utilizing a board PC in the robot because it had a smaller physical footprint and still offered sufficient amounts of computing power. The team selected an IntensePC board computer with an i7 processor and 500GB solid state hard drive which both meet the size constraint, and also provided the power of a good laptop.

The second major source of computing on the diagram is the sub-computer. This computer needed to be able to process requests from the network, while also transmitting lower level commands to the lower level chips and hardware. The lower level requirement was a serious limiter in the number of computers available, but the most popular commercially available products were the Beagle Board and the Raspberry Pi B+. Both were suitable for the job, with the Beagle Board having more recent hardware than the Raspberry Pi B+. While the Beagle Board had this advantage, the Raspberry Pi B+ was chosen for its community support and higher levels of documentation.

The last computer discussed in this section in the tablet peripheral device. The device needed to be a tablet with sufficient battery life to conduct therapy sessions running solely on battery power, lightweight, and enough computing power to run a simple GUI application while processing internet requests. While there are no doubt many that fill these user constraints, we decided to add a constraint of needing to be easy to develop on. This limited the choices to tablets with optional keyboards. With this constraint, we decided to go with the T100 Transformer from Asus.

### 3.4.4 The Operating Systems

The operating systems running on each of the devices greatly affects the functionality of the devices attached, and the capabilities of the programming languages. The operating system is also important to each device because it could greatly affect how quickly and easily someone could expand on the system. Below is a breakdown of every operating system capable device in the system's operating system and why we selected it.

#### 3.4.4.1 The Main Computer

This computer will be running the Windows 7 operating system for this iteration of PABI. Initial plans initially had this computer out of the system, even as far as the cloud. In order to encompass the most amount of common computers, and to utilize large cloud services, the most widespread operating system was used. Windows was also selected because it has better drivers for the devices attached to this device. The team also had access to the tools attached to Visual Studio 2013 to assist in development, and helps package and organize the code for future projects.

#### 3.4.4.2 The Sub-Computer

The sub-computer was not a very powerful device, nor does it need the extras that a full operating system contains. The Raspberry Pi also doesn't have a standard 32 or 64 bit processer, so the natural choice is Linux, more specifically Raspian. The team selected Raspian because it appeared to be the easiest and most documented operating system available, and it was also very compatible with our libraries with the fewest extra installations.

#### 3.4.4.3 The Tablet

In the future, the tablet model could change significantly, so we were less concerned with the operating system on this device. Our concern waned further as this application is destined to be operating system independent on independent packages and languages. Windows 8 was chosen purely because it was on the tablet we purchased by default.

### 3.4.5 The Languages

#### 3.4.5.1 C

C was needed at some points of the project because of lower level embedded computing and OS servicing. C was also LCM capable, allowing our lower level processors to communicate directly over the network. It was also operating system independent, but with varying support and features depending on its host. The most capable and documented of these environments is Linux, so it was used on the sub-computer. While this language could be used on the other devices effectively, it is very time consuming to program the basic structures found in the other languages used. It was used at a minimum in all devices except the sub-computer and lower computers.

#### 3.4.5.2 C++

A further revision of C, C++ has all of the core attributes of C with a library of basic functions built in such as vectors and object oriented capabilities. The threading libraries are well known, well documented, and integrate well with many operating systems. C++ is also compatible with OpenCV, LCM, and many other APIs by default. While not the strictest language to use for the main computer, it was very well known and maintainable for many years to come. Microsoft Visual Studio also provided great support for this language in its projects with may code analysis tools. Given that the main computer will likely use Windows, the support tools make this language the ideal choice to use as a general language for the main computer.

#### 3.4.5.3 Python

Python was used in areas of the project that didn't emphasize efficiency or optimization such as user applications. While this would seem like the language to use for all software components of this project, it was not the best for memory management, and might not perform as expected using vision processing libraries. The Python programming language also does not support threading, a core component of the main computer and sub-computer processes. However, it is a good language to quickly build a GUI that is OS independent and has the capability of large documentation and is easily modifiable. Given that it is also an LCM compatible language, this is a perfect candidate for the Tablet GUI programming.

#### 3.4.5.4 Special Mentions

Other languages were considered to use in PABI, mainly for the tablet application. Java was a first choice because it is a standard for creating applications and it is an LCM library compliant language. We shied away from Java because while this was an alternative platform independent solution, it would have been more time consuming than developing with Python. The language also often goes over revisions and deprecates libraries, making the applications developed stale and unable to be signed by Oracle for release.

### 3.4.6 Logging

The logging of this robot was very verbose in data, but simple in software implementation. Logging was important in this project because it would allow a therapist to look at important areas of the session and determine what is and isn't effective treatment for the patient. Due to the multithreaded, semi-centralized nature of this robot, it is easy to simply give each thread its own handlers to video, audio, and text file writing. The contents of what was logged will be separated into individual files, but a post-processing program can synthesize these contents into a format more readable to the user. The implementation of this algorithm is not contained into this project, but below is the schema of how the data will be presented.

The schema will create a folder named uniquely for the therapy session (likely the date). Inside of this will be a handful of separate files from each thread, each containing different data. Each camera will save a compressed video stream, accompanied by audio from one of the cameras. Questions asked and answered are also to be logged. Data will include a time stamp of the time posed, response time, and if the patient answered correctly. Other things to be added are gaze tracking the patient, motion tracking the patient, and sound graphing the patient.

### 3.4.7 Speech

In order for PABI to communicate with the child, it needed to give auditory cues. The best way of doing this was to utilize a speech API on the main computer. Speech APIs are more flexible to use than raw .wav files and are easier to reprogram as the system matures and expands. The consequence is that the voice is usually restricted to a set of seemingly robotic, generic voices. For this project, we ignored the voice aspect when picking an API and just selected the most convenient and flexible for expansion.

The API used on PABI is the Microsoft SAPI (5.3) that comes standard in any Windows developer SDK. This SDK contains many speech functionalities, including text to speech, and speech to text. The SDK utilized the COM interface in order to send commands to the separate threads internal to it. The speech code did not require construction of its own structure, as the API stood on its own. Through this interface, synchronization was handled for the developer, which also breeds problems in the actual synchronization across the time sensitive threads, mainly in shutting down.

In this revision of PABI, only text to speech was used to convey information. This was easy to use because the PABI therapy database contains the strings and prompts that PABI would give to the patients. The code to perform all speech function was placed on the main computer because the speaker was located on the main computer, and the phrase logic also existed on the main computer.

## 3.4.8 The Code

### 3.4.8.1 Main Computer

The code that runs on this device is computationally intensive, multi-threaded, and preforms most of the hard calculations for PABI. The following sections outline each individual subsystem, following with a wrap-up of everything discussed and how they interact with each other on a lower level. For more information, see Appendix B.

#### 3.4.8.1.1 Networking

The networking API developed for the main computer was critical to keep it as expandable and usable as possible, but not give the user too much trouble to be concerned with the inner workings of the network. The networking API also had to be as lightweight as possible while informing the connected devices of status and state of PABI.

Handling the visibility problem was trivial, seeing that C++ is a fully developed object oriented language. The examples on the LCM GitHub recommend that the handlers for receiving the messages should be defined in classes with their own member variables and attributes. This advice was used in all of the possible messages to receive. Note that all messages, even those which were not needed by the main computer, were accounted for so that no erroneous errors would be unhandled. In order to actually start all of these threads in sync, another class was created in order to ensure appropriate start and termination of the multithreaded libraries. Each of these classes maintain a semaphore to ensure mutual exclusion on getting, and sending data over the wire. The semaphore in the main LCM class is used to exit out of the LCM thread in the form of an inter-process signal. All of the code is located inside of the file handlers.hpp in order to keep code universal across multiple references in future projects.

#### 3.4.8.1.2 Vision

As mentioned above, we decided to use the OpenCV 2.4.9 libraries for our vision needs. The vision processing was very taxing, and required its own share of resources and materials. After completing the LCM library wrapper classes and seeing its success, we decided that doing the same thing would also apply here. A class, referenced in the code as the CVHeader, was developed in order to make the API transparent and the functionality of the separate threads enclosed inside.

The class contained the same overall structure as the LCM API, however the methods inside of the class were relevant for vision processing. This class also maintained a thread handle that allowed the vision processing to be off the main thread so that other processing can be done. The class stored a structure that allowed the main thread to take things such as face poses for head tracking. Semaphores were tied to objects wrapped in accessory methods so that thread lock and race conditions did not exist. The thread encapsulated in this class also logged the image files streamed off of the cameras to be post processed. The methods integrated into the main thread were defined in a header file, but implementation was inside of a .cpp file so that it was swappable as different devices are added or removed in the future.

#### 3.4.8.2 Sub-Computer

The sub-computer was a C programed Raspberry Pi that was programmed to translate the network data from the main computer to the lower level components. This computer was running the Raspian distribution of ARM Linux so that it was easy to program and had a majority of the normal packages that were run on normal Linux distributions.

An API was designed similar to the C++ API for the networking on the chip, however the C API obviously does not have the class aspect of it. This API also automatically allowed the Raspberry Pi to send back status messages to the main computer at the rate of 4Hz. The API sent lock guarded structs over the wire that the user fills through guarded methods. For more information, see Appendix C.

#### 3.4.8.3 Tablet

Most of the work on the tablet application was offloaded to another project due to the massive task involved with the project. While the application itself was not designed or built by the team, the foundation on which it was built was designed and built by the team. This network backend was built almost exactly like the C API for the sub-computer, with the exception that it sent tablet response messages to the main computer at 4Hz, and expected to receive the tablet setup messages at the same rate. Also similar to the C APIs, the get and send message methods build the queue to send into the stream. These actions were also protected by Python lock objects in order to ensure mutual exclusion during critical regions.

The application at the conclusions of this project was intended purely a proof of concept, and was not ready for clinical trials in its state. It contained three images which represented the ABA cards to interact with. After a card was selected, the application would send down the time elapsed since prompt and if the answer was correct for processing by PABI. The main computer would then send a signal back to the tablet with the next cards to display, restarting the process.

# 4 Results

## 4.1 Mechanics

After designing, manufacturing, and assembling the various mechanical components, we conducted a series of tests to compare our original design goals to the performance specifications we established at the outset of the project. The robot was able to be constructed in a robust fashion, and was able to be covered in a soft, visually appealing skin that made the final robot resemble a penguin.

To test the different axes of movement of the robot, we connected each motor to a power supply and determined the maximum range of motion. For the eyes, we determined that the range of motion toward the center of the robot was not as extensive as we had thought based on our CAD model. This deficiency was due to interference of the various fasteners, which were not modeled in CAD, with other structural parts of the robot. The eyes tilted vertically, moving from an angle of 35 degrees down to 12 degrees up. This was a smaller range of motion than our original design, but was limited by other components placed in the head and the head's overall size. The eyelids performed as planned during testing and construction. They were able to both open wide enough for the cameras inside of the eyeballs could see unhindered, and also close fully to simulate blinking. The neck also reflected the correct motion simulated in the CAD model. The neck could pan 90 degrees to either side from center and tilt 30 degrees down and 45 degrees up.

To test the wings, we connected each motor to a power supply and independently tested each axis. Subsequently, we tested moving both axes of each wing together to see how well the wings moved through the range of motion. Due to the complexities in modeling continuum manipulator, we were unable to model the movement of the wings prior to testing them. Individually, each of the four motors controlling the wings was able to move the wing through a full range of motion. Each wing was limited vertically by the ground and the head of the robot and by the body of the robot in the horizontal directions.

## 4.2 Electrical Design

Overall in terms of the electrical design, the electrical systems were able to significantly contribute towards the goals of the project. First of all, the PCBs were successfully designed such that the motors and all sensors could be read and all motors could be supplied with the proper PWM signals and voltages. All of the sensor and motor pins were tested using an oscilloscope, and are known to send signals or receive signals as they should. They also proved to be very space efficient and were able to fit nicely within PABI. The PWM control for PABI worked very well, as all motors could be controlled with a PWM signal that could vary from a 0% (off) to 100% duty cycle. The SPI communication between the Raspberry Pi and the microprocessor was found to work well for the most part, but showed issues when running at the same time as the I2C communication protocols.

## 4.3 Software

The software of the PABI system exceled in operation and the barebones demo showed what the libraries were intended to do. The code presented is able to perform full request and response from the tablet system all the way to the lower processors. The overall architecture was also able to be heavily simplified and compartmentalized into effective, clean code for expansion through APIs developed by the team. Below is a roadmap on how each subsystem was developed and tested in order.

The network structures and pipelines were developed first to ensure proper system wide communication. In order to fully understand what was getting sent without possibly disrupting other network traffic, we created a simple test program to intercept the data and resend as the full system would at that instant. This program did not transmit data over any external networks, but rather read and wrote to its internal UDP loopback. After we confirmed that the library was working and data was being encoded and decoded properly, we began on the Raspberry Pi's receiving code. After a similar test was done in C on the Raspberry Pi to ensure a proper install, the receiving half of the initial code was translated from C++ to C and we then executed an inter-device test. When transmission appeared to be working between the Raspberry Pi and main computer, the same procedure was performed on the tablet. The team encountered challenges when testing the tablet, resulting in setting the UDP streaming flag in the router settings to enable LCM to transmit commands from the main computer to the tablet. Before setting this flag, the tablet would be able to send commands successfully, but not receive any responses from the main computer. After this was completed, we developed full APIs for the system functions in parallel to realize rest of the system architecture. These were tested against each other in toy programs across multiple platforms. We defined toy programs in this use to be simple programs that could exist on the real system, but were too basic or odd to be seriously implemented alone.

With the full network APIs in place, we needed a use case for command control over the network. This came through implementing the OpenCV and CardSelect classes respectively. The OpenCV class sent down control messages to the Raspberry PI, and the CardSelect class would be sending out deck messages to the tablet application through Wi-Fi. Since we had the toy programs already implemented on each target device, all that was required was getting the expected outputs and transmission over the wire. After this development, the main computer's member classes were complete.

With the members complete, the application on the tablet was ready to be implemented into the system, and the Raspberry Pi development was started in parallel. These integrations were trivial, as the groundwork was already in place from the previous steps.

The final step with everything in place was to implement the main thread on the main computer for runtime. Again, this implementation was trivial because everything was built in the previous steps to ensure a speedy development.

# 5 Discussion

## 5.1 Mechanics

### 5.1.1 Wings

The wings presented a unique challenge for the team. Initially intended to be a robust mechanism, the final mechanism proved difficult to work. First, the strings which actuated the wings stretched over time. In addition, we had significant difficulty during assembly tensioning the cables, even before they stretched. Taken together, these deficiencies introduced slop within the mechanism which over time caused the wings to become less responsive and less accurate in their position. In order to address these issues in the future, a more rigid string that would stretch less with use and creating a procedure to tension the strings consistently with the correct amount of tension.

During our initial tests the wings tended to kink near the shoulders, shown below in Figure 60. While we were initially concerned that this would detrimentally affect the final performance of the wings, we found the wings to bend and move in a natural way when they were covered and filled.

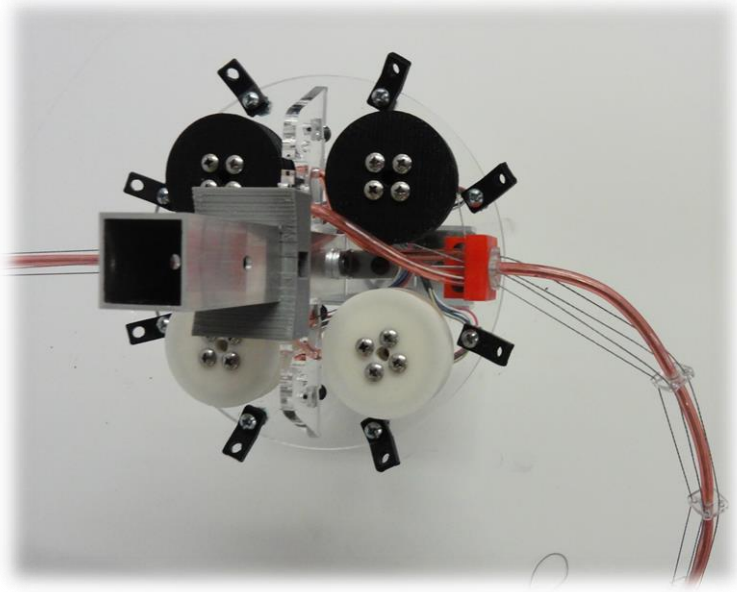


**Figure 60: Bent wing showing kink near shoulders**

### 5.1.2 Eyes

The head and eyes were difficult to execute well due to the number of components and mechanisms that had to be fit into a physically small space. In future iterations, there are several items the team felt could be improved to provide a more robust and serviceable final product. First and foremost, the sheer number of components required a significant amount of time to assemble once all of the components were produced. It was particularly difficult to attach the eyes to the base as well as attach the linear actuator to the motor, as there was not enough space to get a standard screw driver bit onto the screws.

The eyelids also proved difficult to build onto the final eye assembly. In order to create as natural of a look as possible, the space between the eye and the eyelid was minimized. This caused the eyes and eyelids to fit tightly together, but with minimal extra space between each components. In order to add the eyelids to the eye assembly, it was necessary to not only orient the eyes to an extreme orientation, but also slightly deform the eyelids in order to slip them into their final position. While this all the intricate steps required were completed, if the design is going to be produced in mass, designing with the intention of easing the assembly process is paramount, especially for the eyes.

During the initial tests, the linear actuators which powered the eyelids and the pan of the yes were quite stable, however as the mount wore with time, the fit became less precise and the motors began to slide in their mounts. We were able to combat this issue by gluing the motors into the mounts, however this came at the cost of serviceability as removing motor once glued in would damage both the mount and the motor itself. In future iterations, the mounts for the linear actuators should be designed so that a pin can be placed in the rear of the linear actuator to eliminate the need for glue.

Over time, we found that gravity harmed the eye assembly. The primary structural piece was over eight inches long and printed flat on the bed of a Makerbot replicator 2. Due to the thinness of the pieces, which was as little as .01” in places, the piece bent over time. This bending created slop in the eye mechanism, particularly with the eyelids, with the inside of the inner eyelids becoming loose and occasionally coming off of the rod that supported them. While we were able to bend the piece back in to shape to temporarily alieve the bending that had occurred, a more permanent solution is required. A few possible solutions include changing the orientation of the print when it is produces as well as incorporating a reinforcing member of some sort to provide more rigidity through the thinner portions of the piece.

Finally, the mechanism for eye tilt should be improved in future versions. Specifically, the integration of the position sensor is currently very poor, as it is coupled directly to the motor shaft via a 3D printed coupler. It was difficult to manufacture this piece in order to get it to tightly fit around both the round shaft of the potentiometer as well as the D shaft of the motor with minimal mechanical slippage. In the future, this could be solved by using a metal coupler that engages with the shafts using set screws. This would wear better over time as well as ease the assembly process of the head.

### 5.1.3 Overall structure

The biggest challenge that we faced when laying out the components in the body were space constraints, heat that was created by the various computing components, and the manufacturing processes used to create the final product. While all of the components fit inside the body of PABI, the team wanted to fit more battery power within the robot in order to power the motors without being powered from the wall. This would make PABI more mobile and a more effective therapy device since it could interact with the child while the child is carrying PABI.

In several instances, aluminum parts were fastened to one another using bolts and mounting brackets. While these connections generally worked quite well, it was difficult to get all of the slop out of the joints in order to have a stiff, nonmoving joint. In the future, it would be good to weld the joints between the base of the robot, the central spine, and the piece of aluminum channel connecting the

base and the spine together. Welding the parts together would mitigate the chances of the pieces coming loose over time. We did not weld these parts together ourselves due primarily to the lack of welding skill available to us on campus and the high cost of hiring an external welder.

In addition, there were several parts whose manufacture did not support mass production. Most notably, the aesthetic shell of the head was entirely 3D printed. While the use of plastic was appropriate for the material, the amount of time required for manufacture (over 50 hours on a single machine) was a large constraint on our project. In the future, a thinner shell would shorten the amount of time required without affecting the integrity of the final product. At the end of the project, we produced some pieces of the shell with a thickness of 0.25" instead of the original thickness of 0.40". These pieces were successfully integrated into the final robot, proving the feasibility of this design change. Another shortcoming of the aesthetic shell which covered the head was where the two halves met. The upper half of the head, which rotated horizontally, did not have the same profile as the bottom of the head, which did not rotate horizontally. This created some visual holes where a user could potentially see into the inside of the robot. This also created a potential pinch point where someone could get their fingers pinched while the head of the robot turned. To eliminate both of these issues, we created plastic cover that covered the void that was created when the two different pieces of the head rotated.

In the process of manufacturing some of the components, a few parts which were originally intended to be made as one piece were split into multiple pieces in order to speed up production and lower their cost. The most notable example of this was the neck rotation topper. The original part can be seen below in Figure 61. In order to allow the part to be printed on a Makerbot Replicator 2, this part had to be split into two pieces, shown in Figure 62. This allowed the whole assembly to be manufactured in approximately two business days instead of a minimum turnaround of six business days as well as at approximately one tenth the cost.

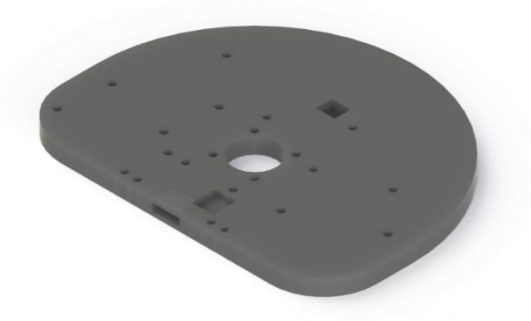


**Figure 61: Original neck rotation topper**

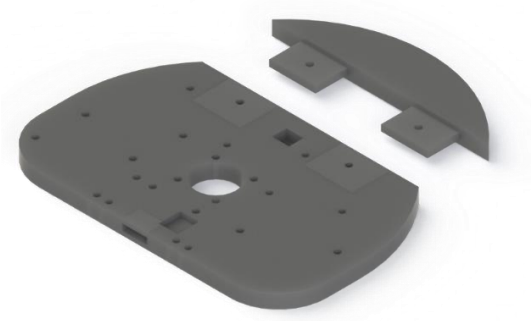


**Figure 62: Modified neck rotation topper in two pieces so it could be manufactured in two piece**

### 5.1.4 Outer Covering

While the cloth and foam that formed the outer covering of PABI was soft and able to create the appearance of a penguin, there were many improvements that could be made in the future. The primary area for improvement is how the shape of the penguin is created. This iteration of PABI was formed using several layers of foam stacked atop one another resulting in a more tiered and layered look rather than smooth curves. In the future, the foam could be integrated into the cloth shell more so that the outer profile was more rounded than it is currently.

## 5.2 Electrical

Overall most of the electrical design for PABI worked well, as discussed previously in the results section. However, there were many areas of the electrical design that could benefit greatly from some changes. This section talks about these potential changes in detail.

### 5.2.1 Component Selection

The following sections discuss the components that were selected for use in this project, as well as recommendations about components for future iterations of PABI.

#### 5.2.1.1 Microprocessor

For the most part, the microprocessor that we used ended up being a good choice. Its 32 bit architecture and 55MHz processing speed made the timing requirements much more relaxed, whereas a slower microprocessor could have had issues with not being able to process the number of input/outputs required. However, the code would have functioned just as well with an 8 bit processor, which would save on cost and have much simpler coding. In addition, the major downfall of this chip was that the ADCs on the microprocessor proved to be extremely difficult, and it would therefore be very beneficial to use a simpler Atmel 8 bit processor with the same number of peripherals. The SPI and I2C communications and the built in quadrature decoders were also very effective, though moving to an 8 bit processor wouldn't negatively affect the performance. Overall, the functionality of this chip left much to be desired, and an 8 bit processor by Atmel would likely work much more nicely.

#### 5.2.1.2 PWM Generators and SPI Level Shifter

The PWM generator provided the needed PWM signals quickly and efficiently to the motor drivers. The major downside of the Adafruit board for the PCA9685 is that it is large and expensive due to having a specialized breakout board. In the future, the PCA9685 and its additional circuitry could be purchased on its own for cheaper than the Adafruit board. This would also save a lot of space on PCBs, going from a board just under two square inches to a chip just under 0.5 square inches. The coding would be similar, as the chip on its own is still meant to output PWM signals, but for LEDs instead of motors. The I2C communication was a good choice because it allowed for both chips to be connected in series and reduce the required wiring on the PCB.

The SPI level shifter also worked as desired, but could be improved for a more space and cost efficient design by integrating the level shift design directly onto the main PCB instead of buying a separate level shifting board to integrate on top of it. The need for this could also be eliminated by using

a microprocessor that can be powered at 3.3V instead of 5V, meaning that the logic would match that of the sub-computer without the need for extra circuitry.

#### 5.2.1.3 Motor Drivers

Overall the motor drivers were a good choice to power the 6V and 12V motors. The built-in over current and over-voltage protection were very useful for making sure that the motors would not break other components when drawing lots of current. The main place for improvement would be designing the motor drivers on the motor board PCBs themselves instead of purchasing board from Pololu. Designing custom motor driver circuitry would save on space and cost, as well as reduce heat dissipation during operation due to having surface mount components instead of through hole components. Another way to improve the system in the future is to connect two motors to each 6V regulator. This was not done initially due to unknowns about the current draw, but the current draw was small enough that each driver could easily drive two motors. A potential drawback is that the cable connecting the motor board to the main board would need to have more connectors, as it would have all of the wiring for two motors and corresponding sensors instead of just one.

#### 5.2.1.4 Connectors

The connectors for the wires proved to be the most inefficient part of the project. While they were small and didn't take up much room on the PCBs, they were very time consuming to assemble and required the use of many very expensive parts and a specialty crimp tool. Once assembled, the wire connectors were very effective at holding the wires in place and not moving. However, due to cost and timing, it would be recommended that in future work that more standard wire connectors be used that could be secured using some other means than a built in locking mechanism, such as some sort of adhesive. In addition, cheaper locking connectors could be purchased, though this would be at the expense of board space.

### 5.2.2 Printed Circuit Board Design

Overall the board design with having a main board that contains the majority of the computing components, a voltage board to centralize the voltage regulators, and a number of smaller motor boards worked well. Having the system split over a number of PCBs allowed the boards to fit efficiently inside of PABI, as well as allow easy access for testing on the boards. This decentralized setup also changes to the boards easier since if something was wrong with the voltage regulation.

### 5.2.3 Embedded Work

Overall the embedded code worked as desired. Both the sub-computer and microprocessor functioned as intended and without any major issues at the conclusion of the project.

#### 5.2.3.1 Raspberry Pi to Microprocessor Protocol

The protocol for communicating desired motor positions and current motor positions and statuses between the sub-computer and the microprocessor worked as intended for the most part. SPI proved to be a good communication system, but information would sometimes be dropped when sending. In the future, a start byte consisting of alternating 0s and 1s could be added as a check to make sure that the packet was accurate before reading it, and a checksum can be utilized. In addition, when

using the SPI and the I2C at the same time, the SPI stops the I2C protocol from working properly. This should be looked into further in future work. Also, the addition of more fields in the communication stream for each motor, such as velocity, would be useful for smoother operation. In addition, a motor status field would be useful to send back to the higher level parts of the system for better error handling and control.

#### 5.2.3.2 Microprocessor

The code on the microprocessor runs efficiently, but lacks integrity in its shared data and file structure. During the course of the project, the microprocessor code was unable to be thoroughly checked for memory leaks, or shared data conflicts. During development tests, no memory leaks or data issues were noticed, but utilizing external code analysis tools for deeper checking is recommended. The code was also structured in an unmaintainable fashion and requires some heavy refactoring for better readability and easier development. This revision of the code did not focus on memory or power efficiency, but optimizing this code could allow a cheaper microprocessor to be used.

#### 5.2.3.3 PWM Motor Control

Using PWM signals for the motor control allowed for easy and low heat motor speed adjustments. The PWM signals are also reliable enabling PABI's different limbs to move at to their desired endpoints smoothly.

#### 5.2.3.4 Future Code

In future iterations of PABI, closed loop control still needs to be implemented. This requires getting the ADC and decoder modules working and implementing a PID controller in the code. A timer is also necessary in order to have consistent closed loop motor control. Current sensing should also be added to the code such that if a current above a specific threshold is measured power to the offending motor should be cut in order to avoid drawing too much current and damaging any components.

## 5.3 Software

While a lot was accomplished and planned for this project, there is much work to be done in order for the PABI system to be complete. The section is outlined into many subsections, and while many of the problems that exist in this system are cross-sectional and multidimensional, they are written under the main system involved.

### 5.3.1 Main Computer

As mentioned before, the main computer is lacking a depth sensor in order to get accurate data. This short coming can be solved by integrating an RGB-D camera. At the conclusion of our project, Intel was planning to release a new model at some point in the second half of 2015 (https://software.intel.com/en-us/realsense/home).

Another recommendation for the code running on the main computer is through the completion of some key functionality. When the project was completed, simple aspects, such as separating the sessions from each other, were not implemented due to time constraints. Other potential functionality includes the ability to intelligently select cards to display or learn from the past experiences of the

session. This is easily adaptable in the CardSelect class, and other classes can easily extend off of the PABI structure to further enhance PABI off of the main thread inside of main.cpp.

A key feature that was missing from the initial specifications was the ability to create and execute high level reinforcement motions to the physical system. It would be nice to have the main computer send broad sweeping commands rather than sending very specific movement movements to the sub-computer and beyond. This was not implemented, and its lack of implementation will make adding intelligence at this level more difficult. We recommend adding a feature such as this before adding some higher intelligence to the main computer.

OpenCV was a valid choice for this project, but its implementation in PABI does not appear to work as well as we had originally hoped. This stems from the inability for the system to receive images from the camera more than twice a second (2Hz). Even after removing the debug properties from the code, removing print statements, and running the OpenCV thread separately, the cameras refreshed at no more than 2Hz. This is insufficient for face tracking and limits future improvements to the face tracking algorithms. While the root cause for this deficiency couldn't be obtained, we speculate that the cameras used could be responsible for the poor performance. The cameras would sometimes become unrecognized by the operating system and cause PABI to fail at startup. There were no alternate cameras available to test with in the timeframe, but further research into this could yield better results for face tracking and frame rates.

LCM was a good choice for implementing a topic-subscriber model inside of PABI, but it came with some high costs. The startup cost for implementing LCM was massive, as it required very specific dependencies and needed different compile flags on the Windows systems. These changes do not affect portability, but took a long time to solve. While the documentation was very detailed and many reports were written on troubleshooting and implementation of LCM, there were still many uncertainties and problems that had to be addressed. The project is still active and under development, and while the revision inside of PABI is stable, we cannot guarantee that an update will be needed in the future. If an update is required, we recommend removing the existing LCM installation and performing a clean install on the newer version. The procedure for installing LCM is located in the documentation of the PABI system.

Lastly, a critical limiting factor in the system is the router and LAN that is created during operation. The router that was used in this iteration of PABI would not work until we flashed the router to the settings already on it. This is unacceptable, and a viable alternative needs to be found in order for full trials and testing to occur. When the network was working, the throughput from the PABI system was not enough to cause any loading or bandwidth issues. This hints at the possibility of running the PABI network queues to send/receive at a speed greater than 4Hz, which is what the system currently runs.

### 5.3.2 Sub-Computer

The sub-computer was mainly feature complete as a pass-through for higher level commands to the microcontrollers in PABI, but it is one of the more interesting places for improvement on the robot. Instead of making this device simply be a translator from the LCM protocol to the SPI protocols on the

microcontrollers, the sub-computer could interpret and execute very common commands, or recover from easily recoverable errors reported from the lower level microcontrollers. This allows the higher level computer to not be burdened with repeating the same commands, or having to account for very basic errors or patterns in the hardware.

Another way to improve the sub-computer code would be to implement a cloud based PABI. Since the commands are not very specific and accept latency, this switch wouldn't be incredibly difficult to the developer. This change would also greatly decrease PABI's power consumption and heat compared with the current model.

### 5.3.3 Tablet

While the tablet application is an essential part of the PABI system, developing it during the course of this project would have been rushed and incomplete. In order to obtain a field ready application, this section of the project was taken up by another student outside of the group while the system was being developed. The application was developed using Python so that it could be built up quickly and effectively without enforcing operating system constraints. While the application works as intended, issues with the wxPython libraries were found, such as long loading times during initialization. The libraries also didn't appear to have the flexibility when designing the UI, requiring the creator to modify the built in libraries. This is obviously undesired, as this greatly limits its ability to be distributed and updated easily. These issues will be worked out in the subsequent doctorate project by the same student.

# 6 References


American Psychiatric Association. (2013). Cautionary statement for forensic use of DSM-5. In Diagnostic and statistical manual of mental disorders (5th ed.). doi: 10.1176/appi.books.9780890425596.dsm01

Bradley, R. (2010). Screening for Autism Spectrum Disorders. *The Journal for Nurse Practitioners, 6*(4), 304-305. doi: http://dx.doi.org/10.1016/j.nurpra.2010.01.002

Breazeal, C., & Scassellati, B. (1999). A context-dependent attention system for a social robot. *rn, 255*, 3.

Brugha, T. S., McManus, S., Bankart, J., & et al. (2011). EPidemiology of autism spectrum disorders in adults in the community in england. *Archives of General Psychiatry, 68*(5), 459-465.

Curtis, A., Shim, J., Gargas, E., Srinivasan, A., & Howard, A. M. (2011). *Dance dance Pleo: developing a low-cost learning robotic dance therapy aid*. Paper presented at the Proceedings of the 10th International Conference on Interaction Design and Children, Ann Arbor, Michigan.

Dautenhahn, K., Fong, T., & Nourbakhsh, I. (2014). A survey of socially interactive robots. from http://www.cs.cmu.edu/~illah/PAPERS/socialroboticssurvey.pdf

Developers, L. (2015). Lightweight Communications and Marshalling (LCM). Retrieved 4/28/2015, 2015, from https://lcm-proj.github.io/

Dickstein-Fischer, L., Alexander, E., Yan, X., Su, H., Harrington, K., & Fischer, G. S. (2011). An Affordable Compact Humanoid Robot for Autism Spectrum Disorder Interventions in Children (pp. 4): Worcester Polytechnic Institute.

Duquette, A., Michaud, F., & Mercier, H. (2008). Exploring the use of a mobile robot as an imitation agent with children with low-functioning autism. *Autonomous Robots, 24*(2), 147-157. doi: 10.1007/s10514-007-9056-5

Fong, T., Nourbakhsh, I., & Dautenhahn, K. (2003). A survey of socially interactive robots. *Robotics and Autonomous Systems, 42*(3–4), 143-166. doi: http://dx.doi.org/10.1016/S0921-8890(02)00372-X

Giullian, N., Ricks, D., Atherton, A., Colton, M., Goodrich, M., & Brinton, B. (2010, 10-13 Oct. 2010). *Detailed requirements for robots in autism therapy.* Paper presented at the Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on.

Granpeesheh, D., Tarbox, J., & Dixon, D. R. (2009). Applied behavior analytic interventions for children with autism: a description and review of treatment research. *Ann Clin Psychiatry, 21*(3), 162-173.

Johal, W., Pesty, S., & Calvary, G. (2014). Expressing Parenting Styles with Companion Robots.

Kidd, C. D., Taggart, W., & Turkle, S. (2006). A sociable robot to encourage social interaction among the elderly. *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, 3972-3976. doi: 10.1109/ROBOT.2006.1642311

Kim, E., Berkovits, L., Bernier, E., Leyzberg, D., Shic, F., Paul, R., & Scassellati, B. (2013). Social Robots as Embedded Reinforcers of Social Behavior in Children with Autism. *Journal of Autism and*

*Developmental Disorders, 43*(5), 1038-1049. doi: 10.1007/s10803-012-1645-2

Kory, J. M., Jeong, S., & Breazeal, C. L. (2013). *Robotic learning companions for early language development*. Paper presented at the Proceedings of the 15th ACM on International conference on multimodal interaction, Sydney, Australia.

Kozima, H., Nakagawa, C., & Yasuda, Y. (2005). Interactive robots for communication-care: a case-study in autism therapy. *Robot and Human Interactive Communication, 2005. ROMAN 2005. IEEE International Workshop on*, 341-346. doi: 10.1109/ROMAN.2005.1513802

Kozima, H., & Yano, H. (2001). *A robot that learns to communicate with human caregivers.* Paper presented at the Proceedings of the international workshop on epigenetic robotics.

Kozima, H., Yasuda, Y., & Nakagawa, C. (2007). Social interaction facilitated by a minimally-designed robot: Findings from longitudinal therapeutic practices for autistic children. *Robot and Human interactive Communication, 2007. RO-MAN 2007. The 16th IEEE International Symposium on*, 599-604. doi: 10.1109/ROMAN.2007.4415155

Lerman, D. C., Iwata, B. A., Hanley, G. P., Madden, G. J., Dube, W. V., Hackenberg, T. D., . . . Lattal, K. A. (2013). Applied behavior analysis *APA handbook of behavior analysis, Vol. 1: Methods and principles* (pp. 81-104). Washington, DC, US: American Psychological Association.

Levin, E. (2012). Software Architecture For A Humanoid Robot For Autism Spectrum Disorder Interventions In Children (pp. 45): Worcester Polytechnic Institute.

Levy, S. E., Mandell, D. S., & Schultz, R. T. Autism. *The Lancet, 374*(9701), 1627-1638. doi: http://dx.doi.org/10.1016/S0140-6736(09)61376-3

LLC, I. (2014). Popchillia's World. Retrieved 9/26/2014, 2014

Lovaas, O. I. (1987). Behavioral treatment and normal educational and intellectual functioning in young autistic children. *Journal of Consulting and Clinical Psychology, 55*(1), 3-9. doi: 10.1037/0022-006X.55.1.3

NetMarketShare. (2015). Desktop Top Operating System Share Trend. Retrieved 4/28/2015, 2015, from http://www.netmarketshare.com/

Odom, S. L., Thompson, J. L., Hedges, S., Boyd, B. A., Dykstra, J. R., Duda, M. A., . . . Bord, A. (2014). Technology-aided interventions and instruction for adolescents with autism spectrum disorder. *Journal of autism and developmental disorders*, 1-15.

Qidwai, U., Shakir, M., & Connor, O. B. (2013). Robotic toys for autistic children: Innovative tools for teaching and treatment. *GCC Conference and Exhibition (GCC), 2013 7th IEEE*, 188-192. doi: 10.1109/IEEEGCC.2013.6705773

Rivera, A. M. (2013). PABI©'s Design & Development (pp. 74): Worcester Polytechnic Institute.

Robins, B., Dautenhahn, K., Te Boekhorst, R., & Billard, A. (2005a). Robotic assistants in therapy and education of children with autism: can a small humanoid robot help encourage social interaction skills? *Universal Access in the Information Society, 4*(2), 105-120.

Robins, B., Dautenhahn, K., Te Boekhorst, R., & Billard, A. (2005b). Robotic assistants in therapy and education of children with autism: can a small humanoid robot help encourage social interaction skills? *Universal Access in the Information Society, 4*(2), 105-120.

Scassellati, B., Henny, A., & Matarić, M. (2012). Robots for Use in Autism Research. *Annual Review of Biomedical Engineering, 14*(1), 275-294. doi: 10.1146/annurev-bioeng-071811-150036

Shamsuddin, S., Yussof, H., Ismail, L. I., Mohamed, S., Hanapiah, F. A., & Zahari, N. I. (2012). Initial Response in HRI- a Case Study on Evaluation of Child with Autism Spectrum Disorders Interacting with a Humanoid Robot NAO. *Procedia Engineering, 41*(0), 1448-1455. doi: http://dx.doi.org/10.1016/j.proeng.2012.07.334

Short, E., Swift-Spong, K., Greczek, J., Ramachandran, A., Litoiu, A., Corina Grigore, E., . . . Scassellati, B. (2014). How to Train Your DragonBot: Socially Assistive Robots for Teaching Children About Nutrition Through Play. *23rd IEEE Symposium on Robot and Human Interactive Communication*(August).

Simpson, R. L. (2001). ABA and Students with Autism Spectrum Disorders: Issues and Considerations for Effective Practice. *Focus on Autism and Other Developmental Disabilities, 16*(2), 68-71. doi: 10.1177/108835760101600202

Tapus, A., Maja, M., & Scassellatti, B. (2007). The grand challenges in socially assistive robotics. *IEEE Robotics and Automation Magazine, 14*(1).

Valicenti-McDermott, M., Hottinger, K., Seijo, R., & Shulman, L. (2012). Age at Diagnosis of Autism Spectrum Disorders. *The Journal of Pediatrics, 161*(3), 554-556. doi: http://dx.doi.org/10.1016/j.jpeds.2012.05.012

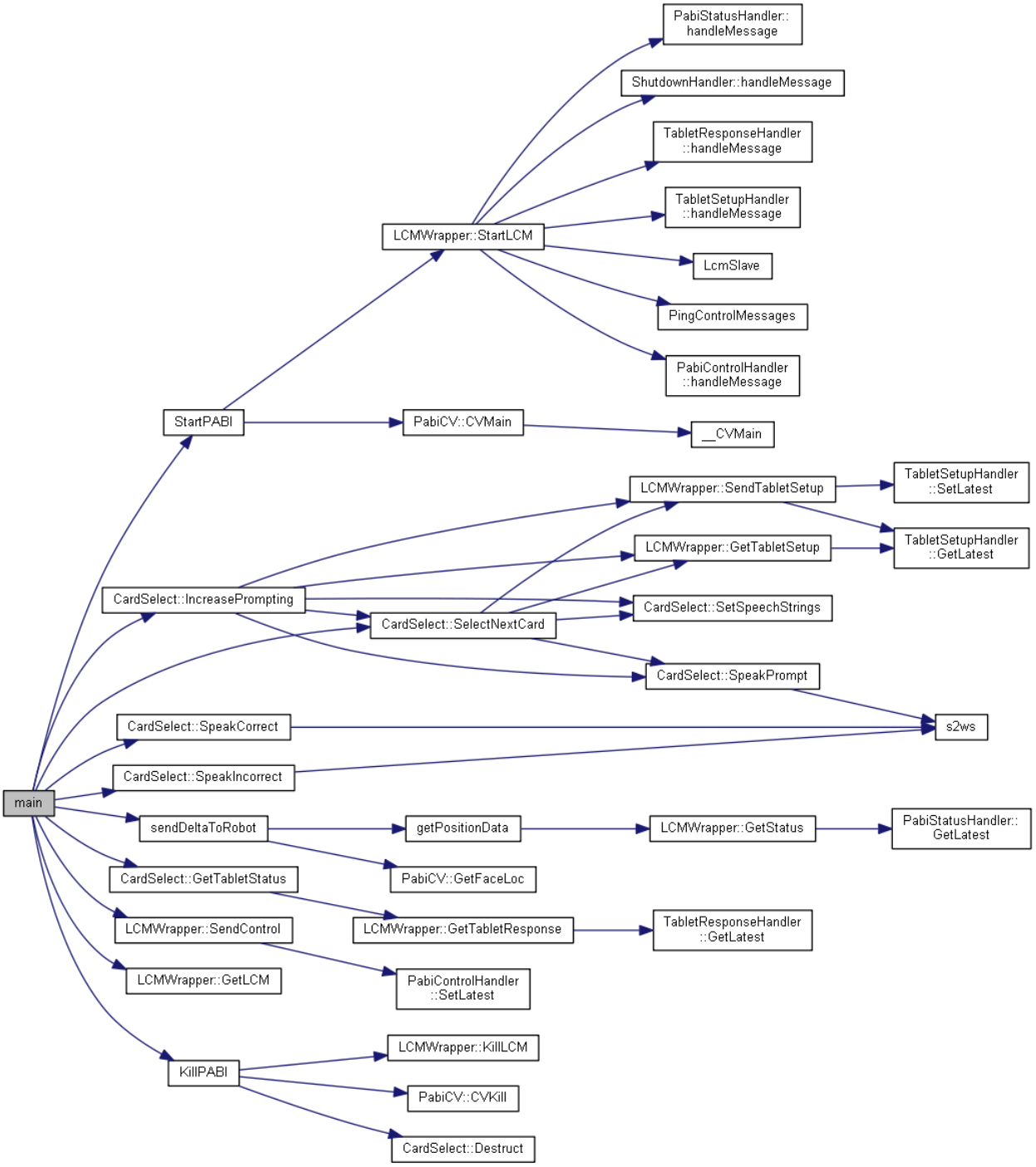
Volkmar, F. R., & Pauls, D. (2003). Autism. *The Lancet, 362*(9390), 1133-1141. doi: http://dx.doi.org/10.1016/S0140-6736(03)14471-6

Wainer, J., Dautenhahn, K., Robins, B., & Amirabdollahian, F. (2010). Collaborating with Kaspar: Using an autonomous humanoid robot to foster cooperative dyadic play among children with autism. *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*, 631-638. doi: 10.1109/ICHR.2010.5686346

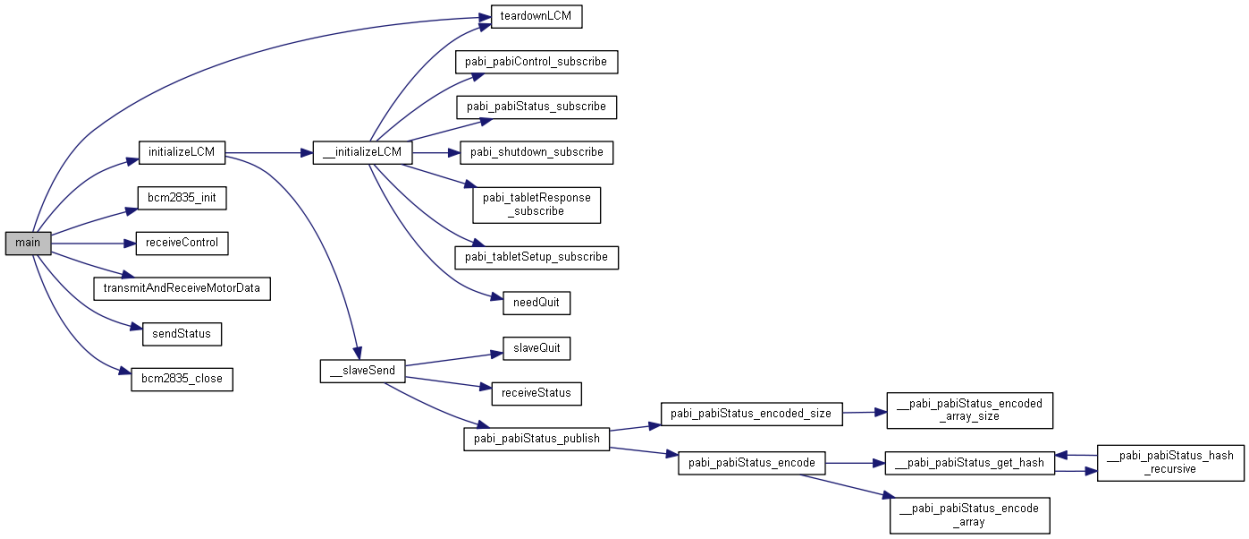
# Appendix A: Custom Cable Production Information

| Cable Description | Quantity | # Wires | Length (inches) |
|---|---|---|---|
| Main Board to Fan PCB | 1 | 5 | 10 |
| Main Board to Neck Tilt PCB | 1 | 9 | 8 |
| Main Board to Eyelids PCB | 2 | 7 | 38 |
| Main Board to Eye Pan PCBs | 2 | 7 | 38 |
| Main Board to Beak PCB | 1 | 7 | 38 |
| Main Board to Neck Pan PCB | 1 | 9 | 8 |
| Main Board to Wings PCB | 4 | 7 | 14 |
| Main Board to Eye Tilt PCB | 1 | 7 | 8 |
| Main Board to Voltage Board | 1 | 5 | 6 |
| Main Board SPI to Raspberry Pi | 1 | 5 | 10 |
| Main Board to Voltage Board | 13 | 2 | 2.5 |
| Wing Motor Extension | 4 | 6 | 8 |
| Wing Potentiometer Extension | 4 | 3 | 15 |
| Neck Pan Extension | 1 | 6 | 36 |
| Eye Tilt Potentiometer Extension | 1 | 3 | 40 |
| Neck Tilt Motor Extension | 1 | 6 | 36 |
| Eye Tilt Motor Extension | 1 | 2 | 40 |

# Appendix B: Main Computer Flow Diagram



# Appendix C: Sub-Computer Flow Diagram



# Appendix D: Custom PCB Schematics

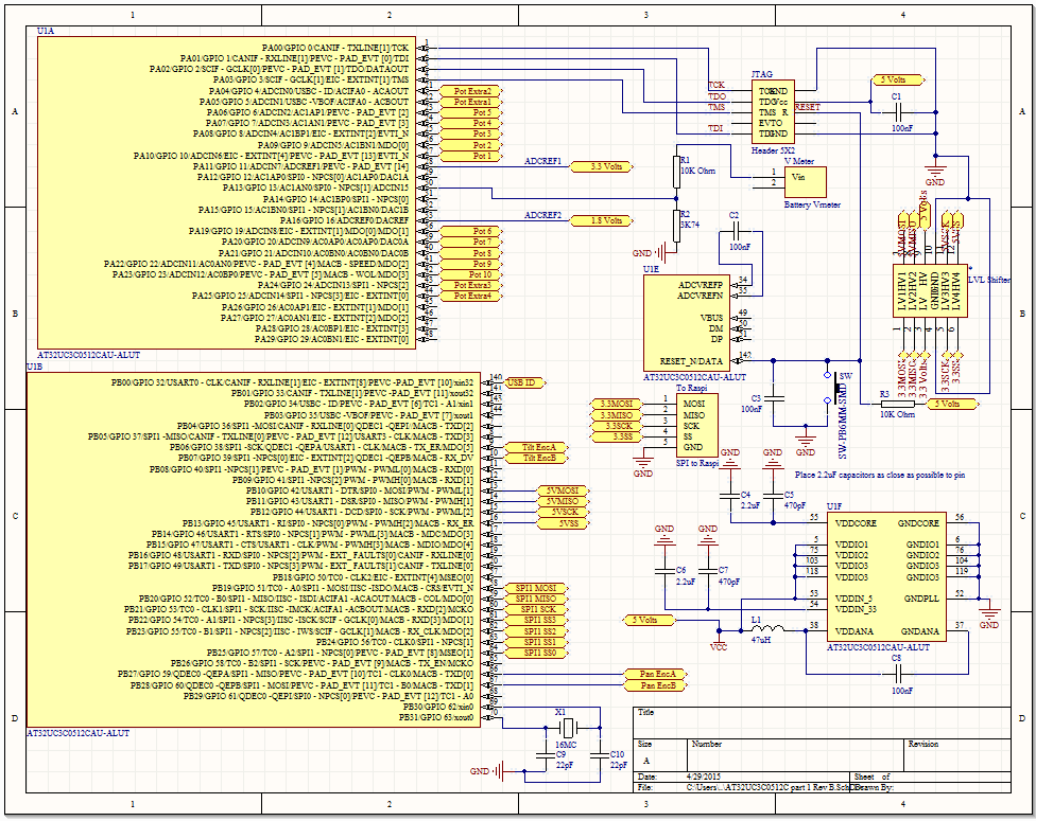


**Figure 63: Main board schematic - part 1 of 4**

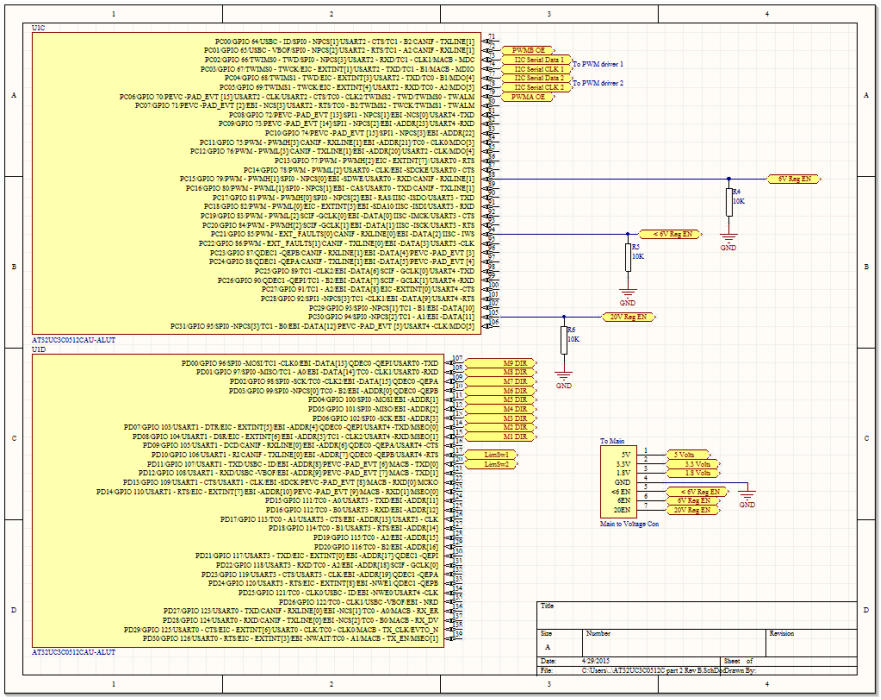


**Figure 64: Main board schematic - part 2 of 4**

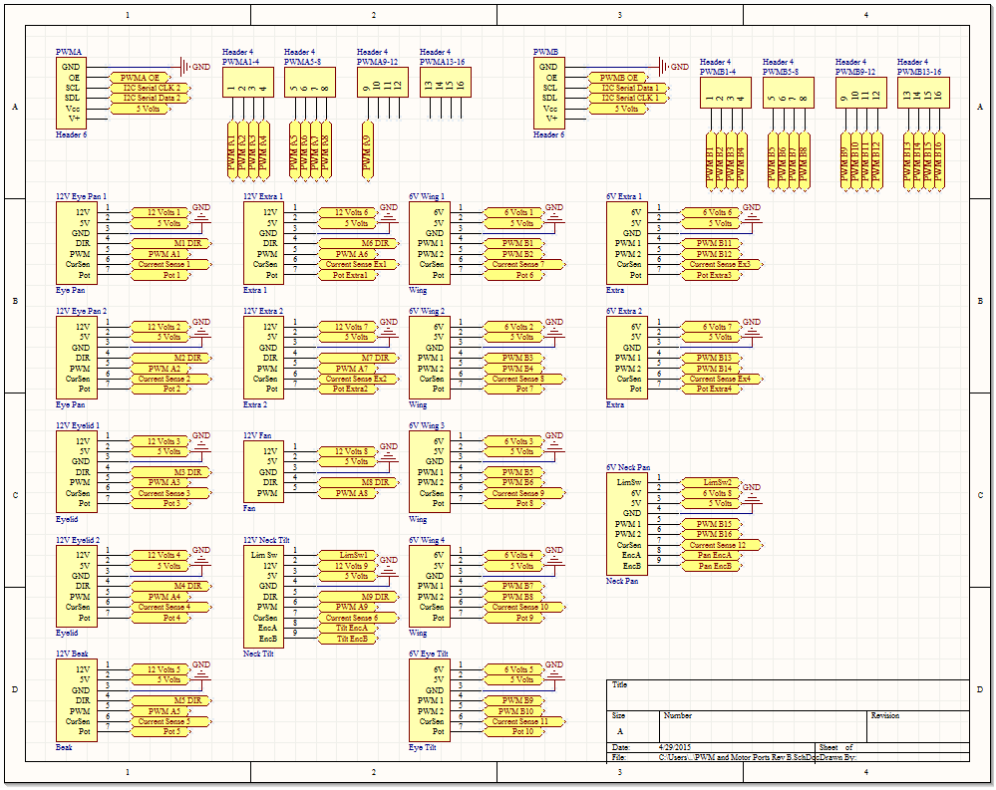


**Figure 65: Main board schematic - part 3 of 4**

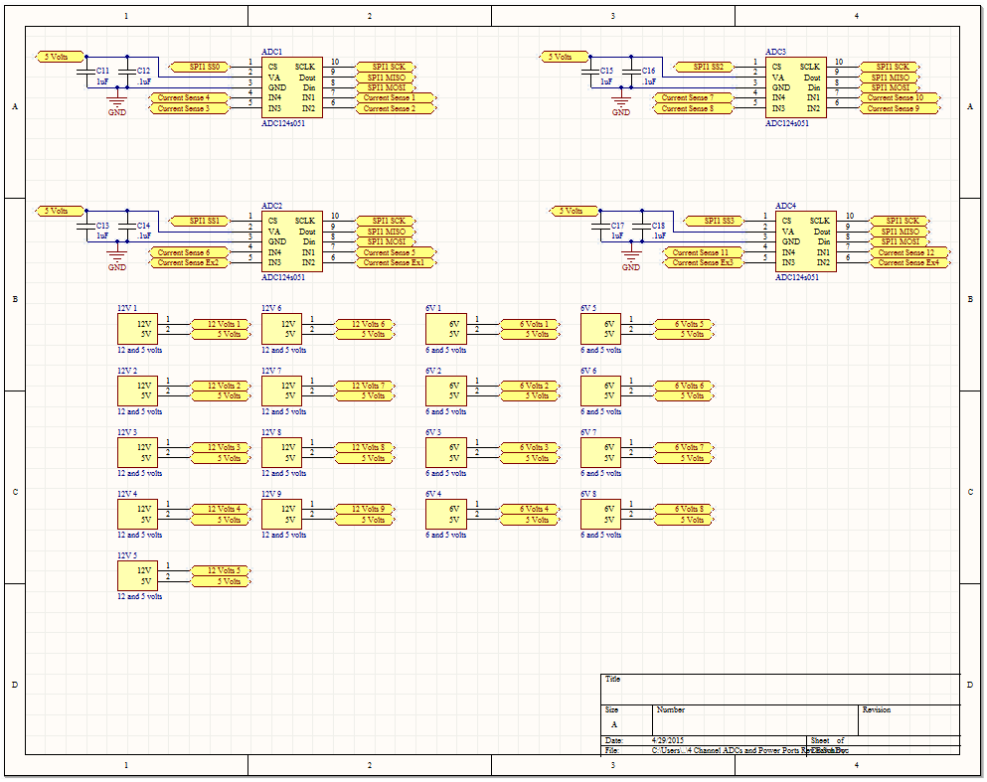


**Figure 66: Main board schematic - part 4 of 4**

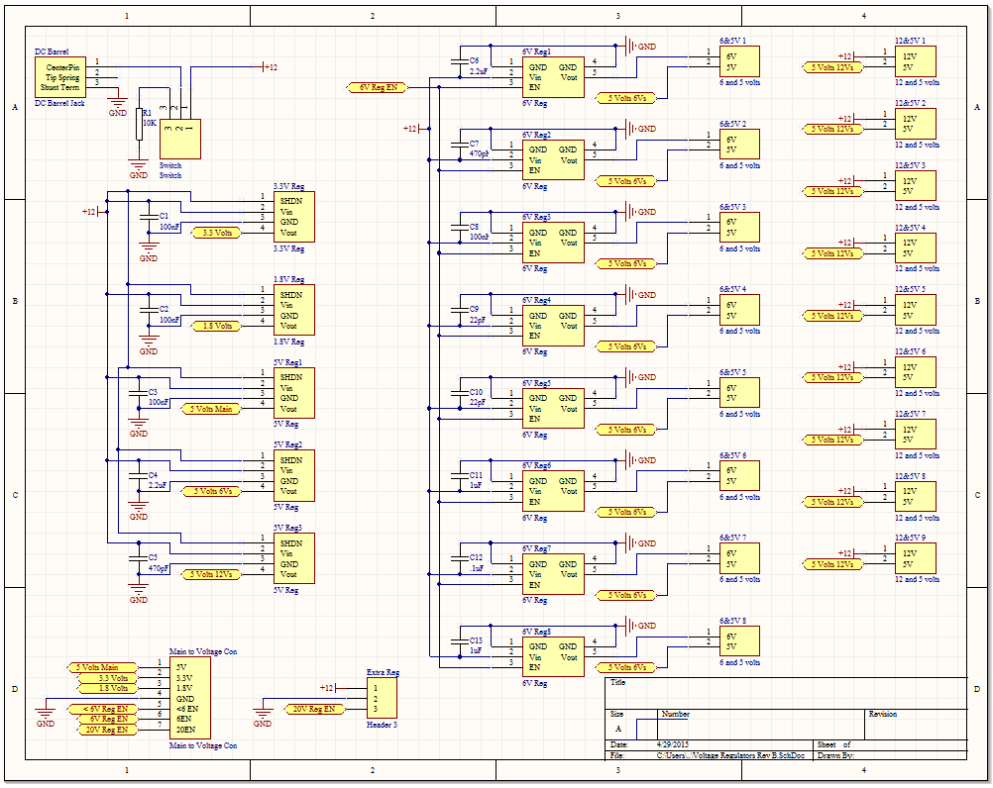



**Figure 67: Voltage board schematic**

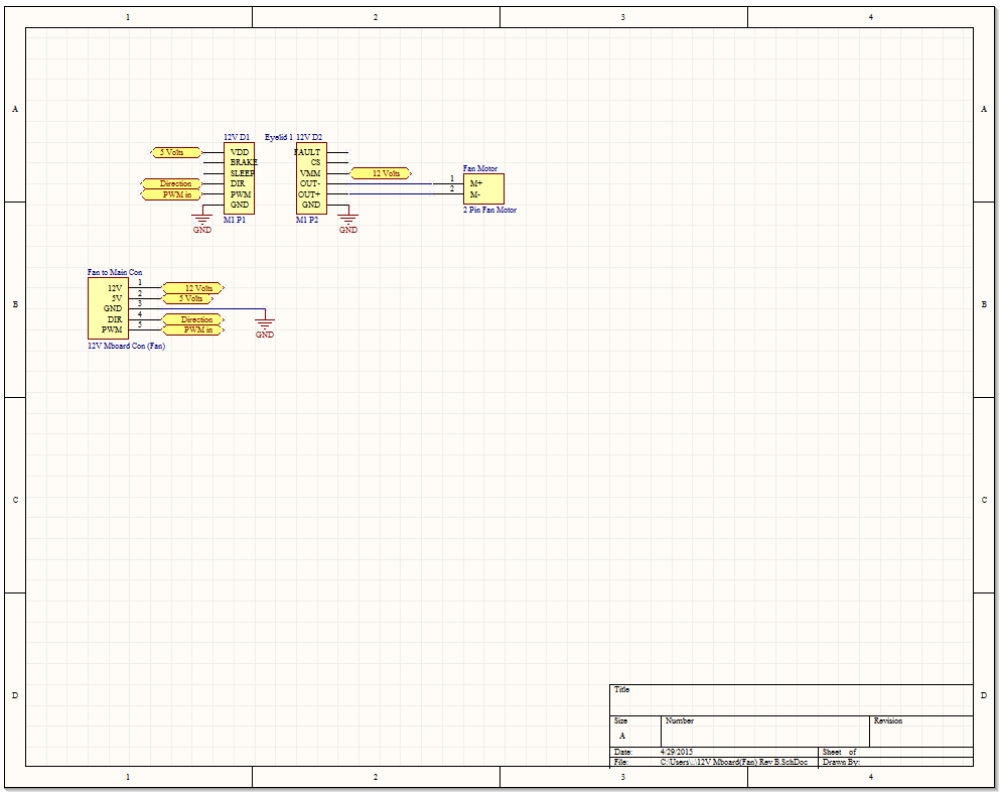



**Figure 68: Schematic for 12V fan board**

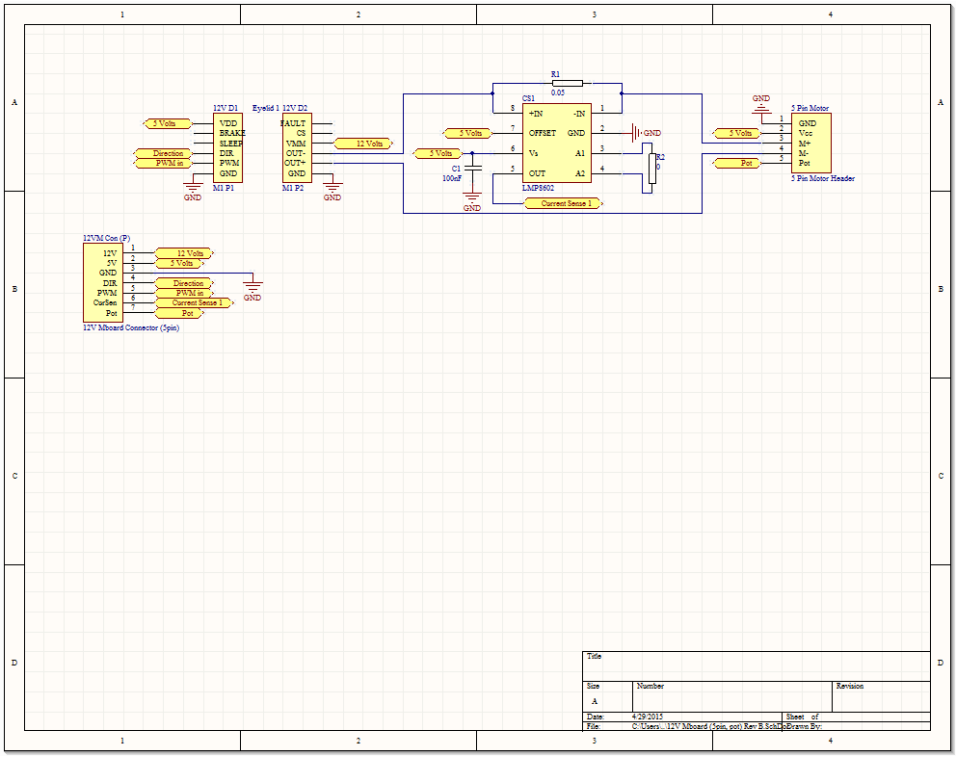



**Figure 69: Schematic for 12V potentiometer board**

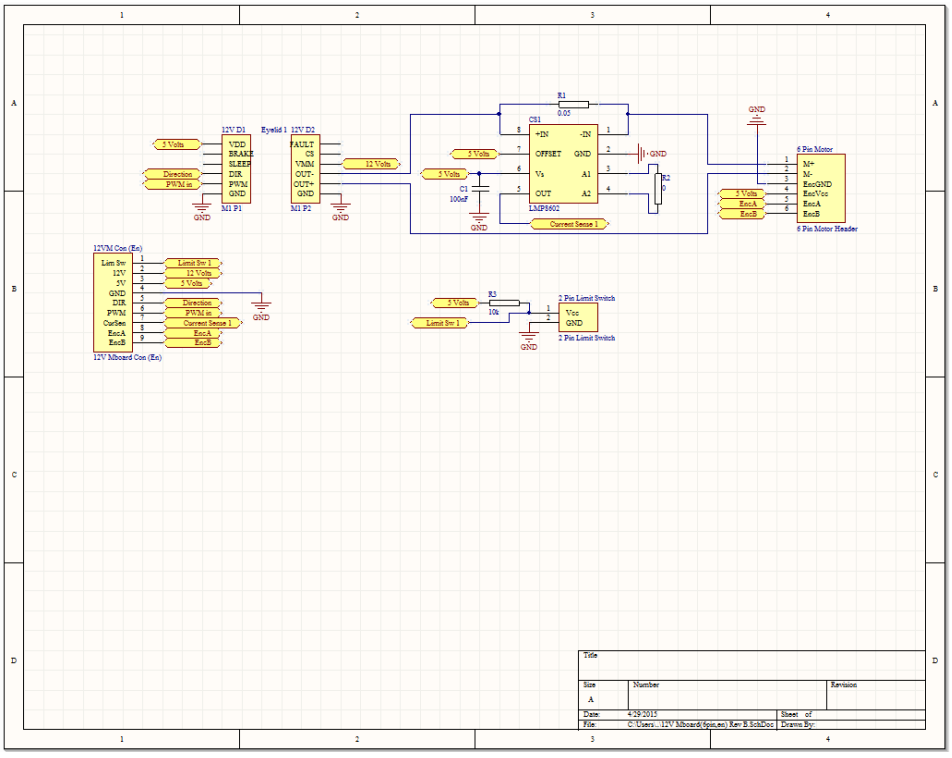



**Figure 70: Schmatic for 12V encoder board**

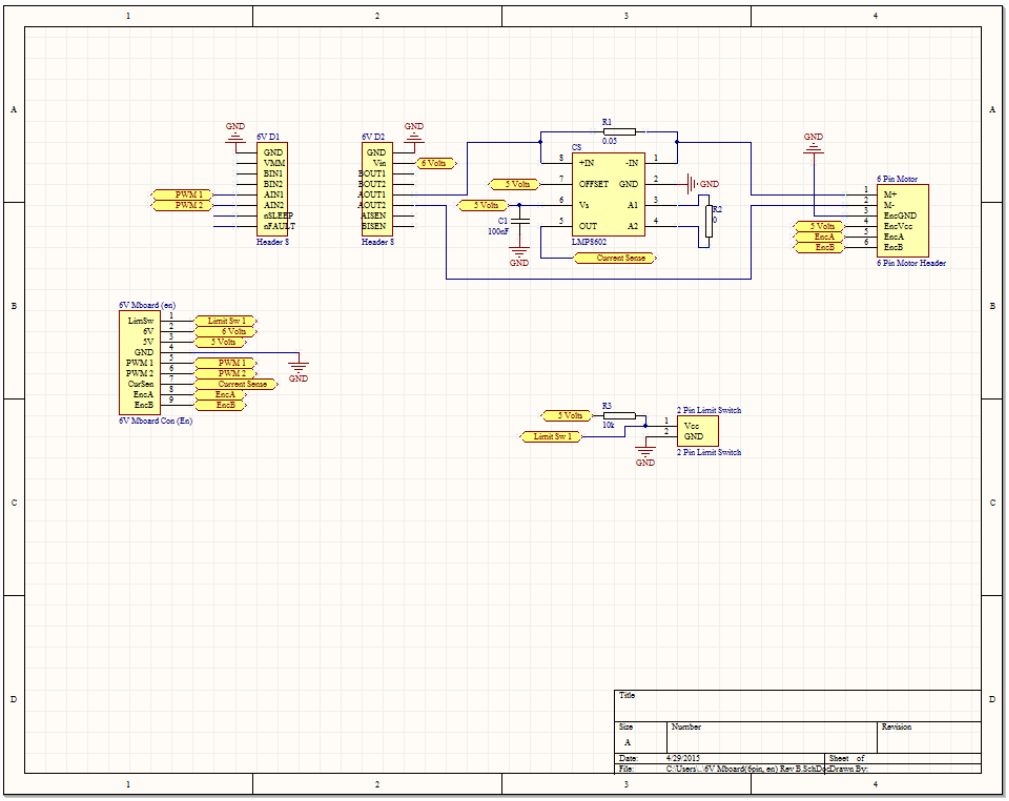



**Figure 71: Schematic for 6V encoder board**

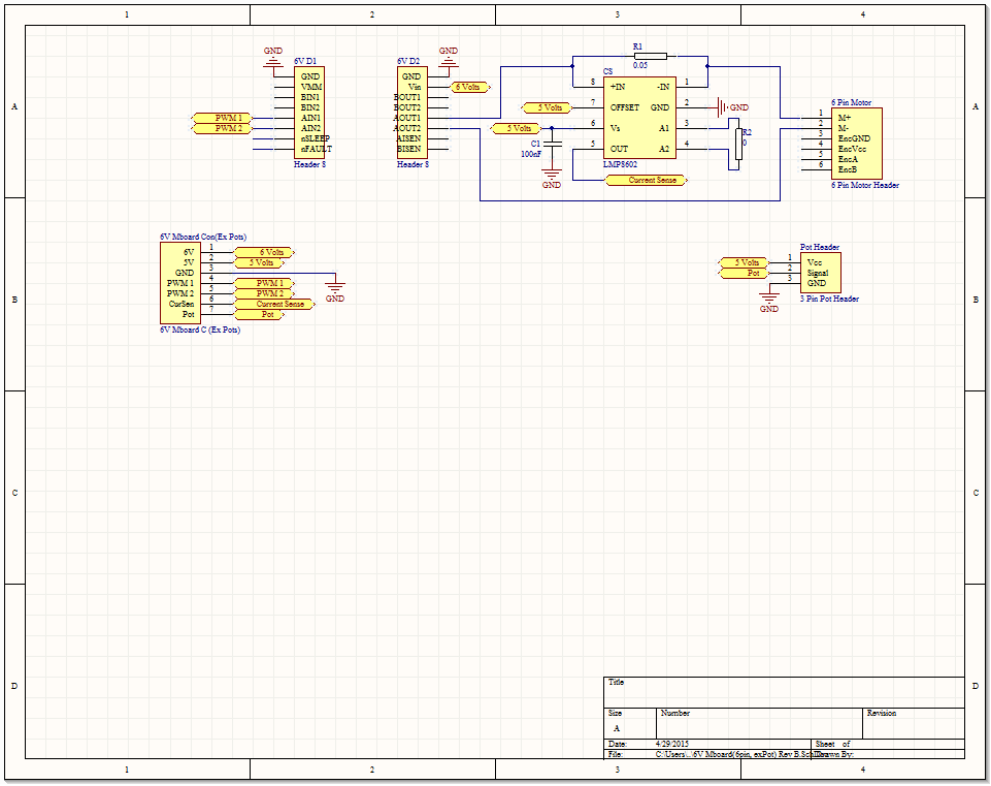



**Figure 72: Schematic for 6V potentiometer board**