Wireless Robotic Arm

Team Fort Teen
Ian Hogan (hoganic@bu.edu), Josh Wildey (jwildey@bu.edu)
https://github.com/jwildey/ec535Project

Abstract - *This report describes the design and implementation of a wireless robotic arm. A prototype of this arm was made that could be used in a variety of applications such as surgery or explosive ordinance disposal. The intent of the project was to have the robotic arm controlled by a wearable glove, powered by a Gumstix computer, that would have an accelerometer and flex sensors. Due to hardware constraints and time, the glove was simplified to a handheld controller which is then used to send signals to a Raspberry Pi connected to an Arduino microcontroller that controls the motion of the robotic arm. The handheld controller utilizes general purpose input/output pins as digital signals which are interpreted into commands which are then sent to the robotic arm using a combination of Bluetooth connection and USB serial connection.*

1. Introduction

The topic of the project is to be able to wirelessly control a robotic arm in a manner that the concept can be used for fine motor skill equivalent. Fine motor skills incorporates the smaller control and dexterity muscles of the human body. In order to simulate this with robotics the robot, using hardware and code, must operate in a real time setting and be able to make small adjustments set by the user.

The human body can perform various magnificent operations that have still not been matched by robotics and computer systems, yet. However the body degrades over time losing accuracy and dexterity in movement and is also not expendable like a machine. Examples of jobs and occupations where these attributes are essential range from surgery to explosive ordnance disposal (EOD) to prosthetics for lost limbs.

In the case of surgery this is what our project more so focuses on, with the goal to perform a task that requires fine motor skills and is even a test given to those with new prosthetic arms/hands. The greatest surgical doctors are noted for their fine skills with their tools, taught to perform at extremely high levels of precision and dexterity. Over time these skills degrade while the mind of the doctor is still capable of performing surgeries. With the assistance of high precision robotics and embedded systems these high precision surgeries can be performed at a level never seen before by human hands.

At lower, proof of concept, models the best way to increase accuracy and precision for a robotic arm is to use stepper motors and encoders to relay back information concerning speed and location of the "limb". However to increase physical accuracy of the system to the nano or micrometer level piezo actuators can also be used. Accuracy can be added to the controller via the speed and movement controls sent to the motors. For increased accuracy the speed can be

lowered while also reducing the time that the motor movement is active, until the next data packet. To decrease communication delays the sampling rate at which the data packets are sent from the controller can be increased causing a more real time reaction to the system. All of these methods are used in our project to accomplish fine motor skills.

Basic functionality of the system includes bluetooth communication to properly relay data reliably from the controller to the robotic arm, be able to control all 4 motors in a forward and backwards orientation while also controlling their speed of movement, and be able to relay all controls correctly, in a specific order, from the controller to the robotic arm. Once basic functionality was accomplished finer motor skills were fine tuned by increasing sampling and communications speeds of the bluetooth microprocessors while also decreasing the time that the robotic arm takes to complete one designated movement. By doing this we are able to visually keep the robots movements, in reaction to the controller, in real time while also increasing accuracy of every movement.

2. Design Overview

The final design of the wireless robotic arm consists of two major components that communicate via a Bluetooth connection. The first component of the design is the one controlling the robotic arm whose block diagram is shown in Figure 1 below. To control the robotic arm, it uses H-bridge ICs to control DC motors, using two DIOs signals and one PWM signal per motor, that actually provide the motion control for the arm. Potentiometers were installed to provide feedback of the arm's position since the motors provided with the arm were simple DC motors without encoders to provide motor axis positions. A servo motor serves as the control mechanism for the gripper used on the arm which takes a standard PWM signal from the Arduino. The Arduino receives commands through a serial connection made via USB with a Raspberry Pi. The Raspberry Pi essentially acts as a command router. It receives commands from the Gumstix powered controller via a Bluetooth connection and forwards the commands on to the Arduino through the USB serial connection.

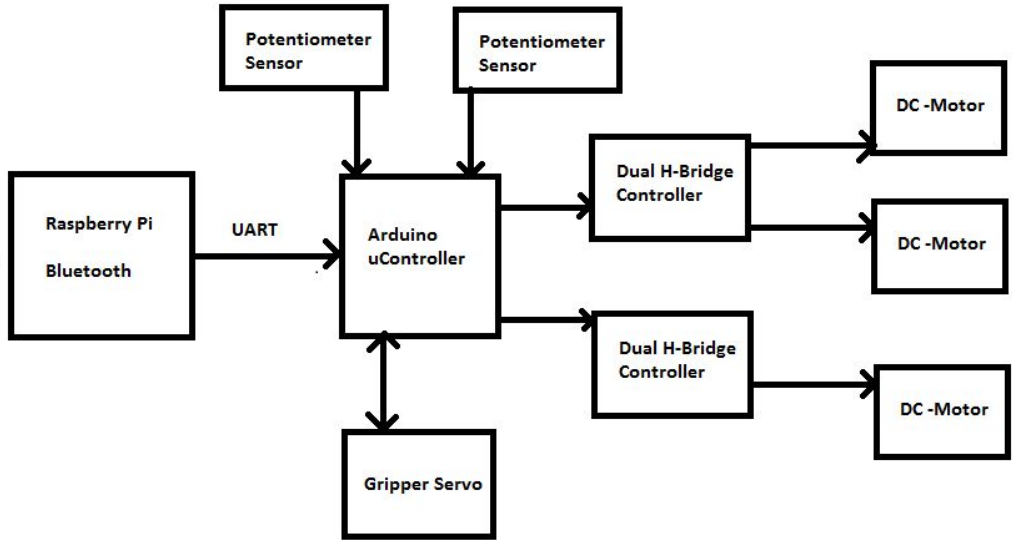

Figure 1 - Final Robotic Arm Block Diagram

The second major component of this design is the controller which uses a Gumstix to read inputs from sensors, translate them into commands and sends the commands to the Raspberry Pi via the Bluetooth connection. A functional block diagram is shown below in Figure 2. The sensors of the final design ended up being digital input/output (DIO) lines that were controlled by push buttons or dual threshold comparators connected to an analog joystick. A kernel driver was written to interface to these DIO lines and translate them to commands. A user space application was also written to interface to the kernel driver and to communicate with the Raspberry Pi via Bluetooth.

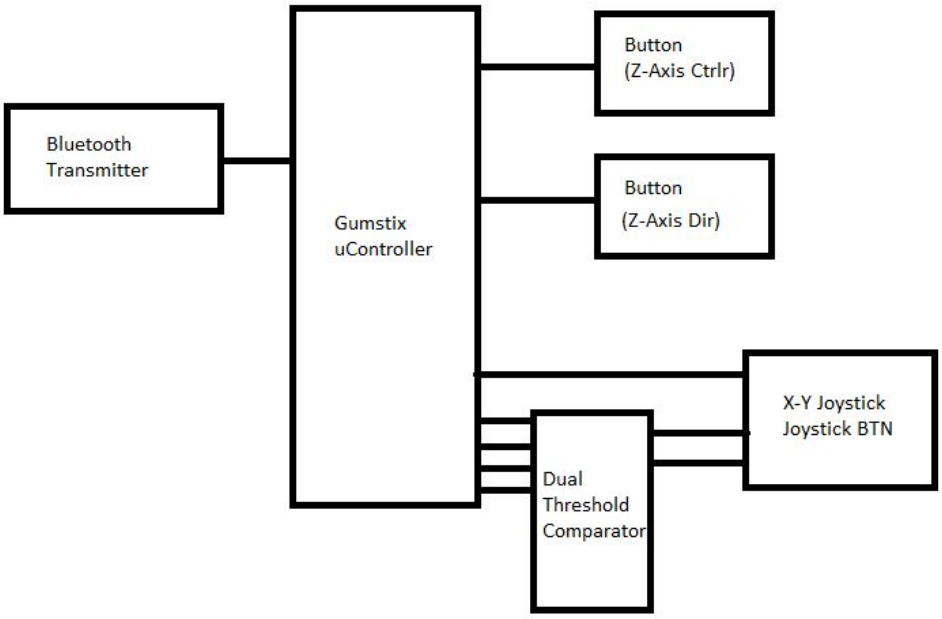

Figure 2 - Final Controller Block Diagram

All of the circuitry for the threshold comparators was designed and built by Ian. Josh and Ian both took part in writing the kernel driver for the Gumstix controller. Ian defined all the GPIO to be utilized by the inputs and the setup of the recurring kernel timer. Josh wrote the skeleton for the file and wrote the init, exit, read and write functions to interface with user space. Josh also wrote the user space level program that read the device file, created the Bluetooth connection and wrote the commands to the Bluetooth connection to be received by the Raspberry Pi.

Josh wrote the routing software used on the Raspberry Pi to receive packets via a Bluetooth connection from the Gumstix. It then adds a header and writes those packets to the Arduino via a serial connection over USB.

Ian wrote all the software for the Arduino which receives the packets over the serial USB connection and then writes those commands to the motor controllers appropriately. Ian also was responsible for the wiring of the robotic arm to the Arduino.

Josh also wrote code whose purpose was to communicate to an accelerometer via I^2C, but in the interest of time, this effort was set aside to create a working prototype. Josh also did most of the management of the Github repository.

3. Project Details

a. Preliminary Glove Design

The original design of this wireless robotic arm involved a wearable glove that had an accelerometer and flex sensors on it. The data accelerometer was to be used to determine the movement of the robotic arm and the flex sensors were going to control the state of the gripper at the end of the robotic arm. The functional block diagram of this original design is shown below in Figure 3.

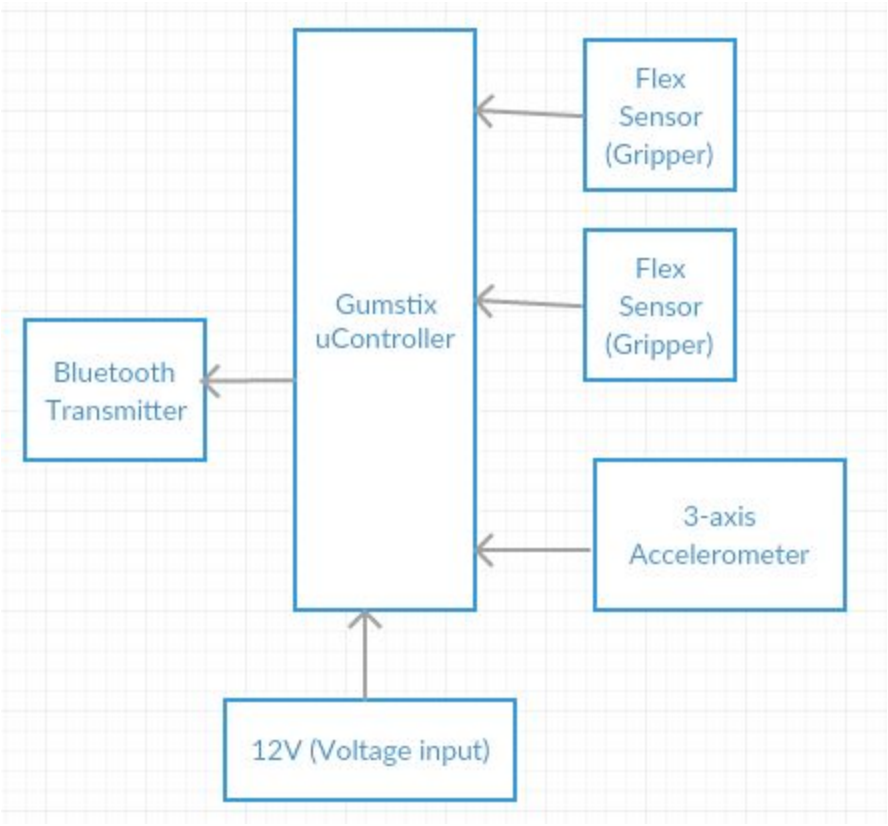

Figure 3 - Original Glove Block Diagram

The accelerometer only has one interface which is I^2C. At first a user level program was written in an attempt to utilize the existing I^2C kernel driver included with the OE software package provided with the Gumstix. The first attempt with this approach opened the '/dev/i2c-0' device file and used the IOCTL to assign the address of the accelerometer. The read and write functions were then used to read and write data. This approach had poor results in that nothing was either written or read from the ports.

A second user space level approach was taken similar to the one described above. The difference in this approach was how it used the device file. Instead of using the read and write functions of the device file, the IOCTL function was used in attempt to directly

control what was written and read from the I^2C port. Code was taken directly from the source code of BusyBox and ported over into this user space program and this too had similar results to the first approach.

Lastly, an attempt was made to create our own kernel driver for the I^2C interface. Instructions were followed in section 9.1 of the Gumstix processor developer's manual [1]. This involved setting up the GPIO pins to their appropriate alternative functions, enabling the correct clocks and enabling/disabling desired interrupt functions. The driver was set up to not use interrupts and to use a polling method of the I^2C status registers to tell when data transfer is complete. The desired data was loaded into the data transfer registers and the correct bits were set/clear according to the manual. Unfortunately this effort also had poor results and this I^2C effort was sidelined in an effort to get a functioning prototype.

The flex sensors that were planned to control the gripper were to use an analog input utilizing a basic voltage divider, however the Gumstix controller does not have any ADCs and the ADCs we ordered communicated via a serial peripheral interface (SPI). Due to the time spent on trying to get I2C to work, we opted to just use a button to control the gripper.

Due to these reasons, the glove design was simplified to just using DIO inputs to generate the commands of the glove.

b. Preliminary Arm Design

The original design of the robotic arm was to build a robotic arm with three degrees of freedom while using servo motors, seen below Figure 4. Servos were desired due to their simplicity to use and to read back position of the robotic arm. In order to save time and to begin programming the robotic arm as soon as possible we decided to use an OWI Robotic Arm provided to us. The downside to this robotic arm was that the motor controller (located on the robotic arm) could not be used and the motors are DC-motors.

DC motors lack accuracy and feedback which is desired in our project since the objective of the robotic arm is to perform fine motor skills. We solved or mitigated these issues with hardware and software solutions. With hardware we added two potentiometers to the joints of the elbow and the wrist motors, idea originated from instructables [2]. This allows us to read, through analog inputs, the positions of the arms. Step motors can stop at fixed pivot locations for high accuracy movements and control, on the other hand DC motors carry acceleration and inertia once you want them to be halted. To minimize this effect we lowered the speed of the motors using PWM controls with a dual H-Bridge motor controller as well as decreasing the time that the motor is activated while increasing sampling time to read control data. This results in the motors running for a

shorter time, reducing their inertia.

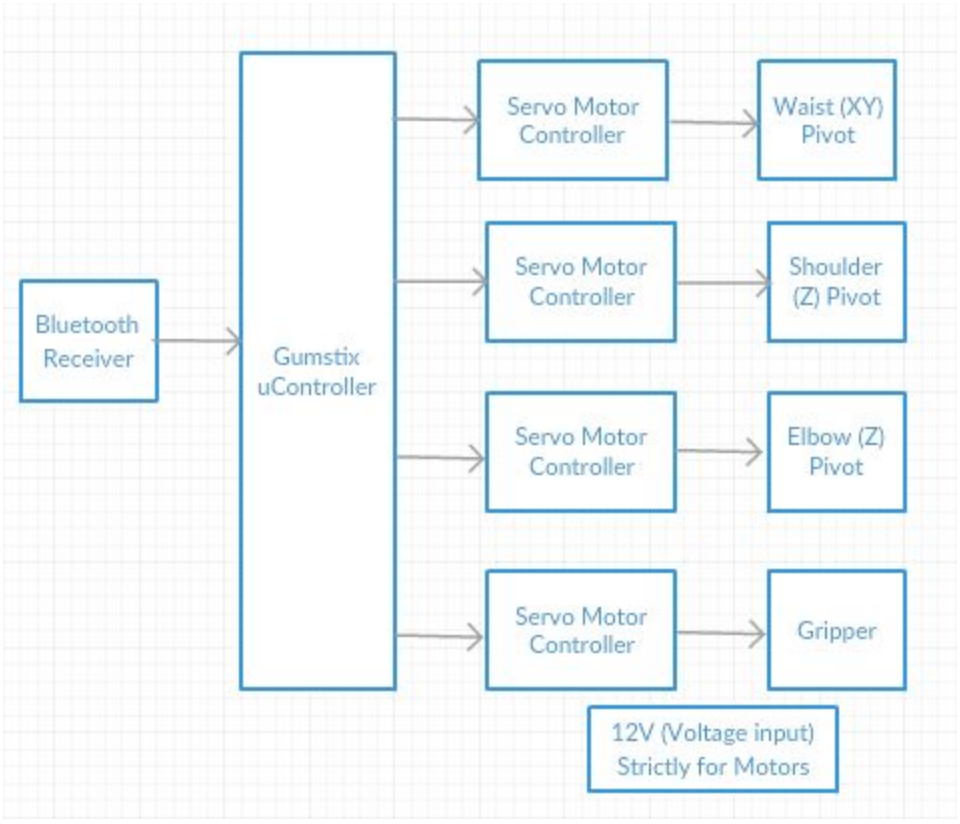

Figure 4 - Original Robotic Arm Block Diagram

Due to the required use of four PWM channels and the gumstix only having 2 PWM pins design of the code had to be moved over to the arduino. To connect over bluetooth the arduino now required a bluetooth module to communicate to the other gumstix controller. We decided to use an HC-05 bluetooth module that is capable of both slave and master configurations. Unfortunately the gumstix was unable to connect to the HC-05 to do anything besides SSH onto the gumstix and take the board over. Whether the gumstix was the slave or the master it would not connect while sending or receive data from the module. At one point the gumstix was able to pair with the HC-05, however the gumstix would drop its connection upon the HC-05 turning off, and not being able to reconnect to the HC-05 without the module entering setup mode. To finally avoid this issue we treat a raspberry pi as a data router to receive bluetooth information from the gumstix controller while relaying this data over serial UART to the arduino.

Due to the above issues and resulting solutions and mitigations the circuit and robotic arm design has changed to that which can be seen in section h below.

c. Controller Circuitry

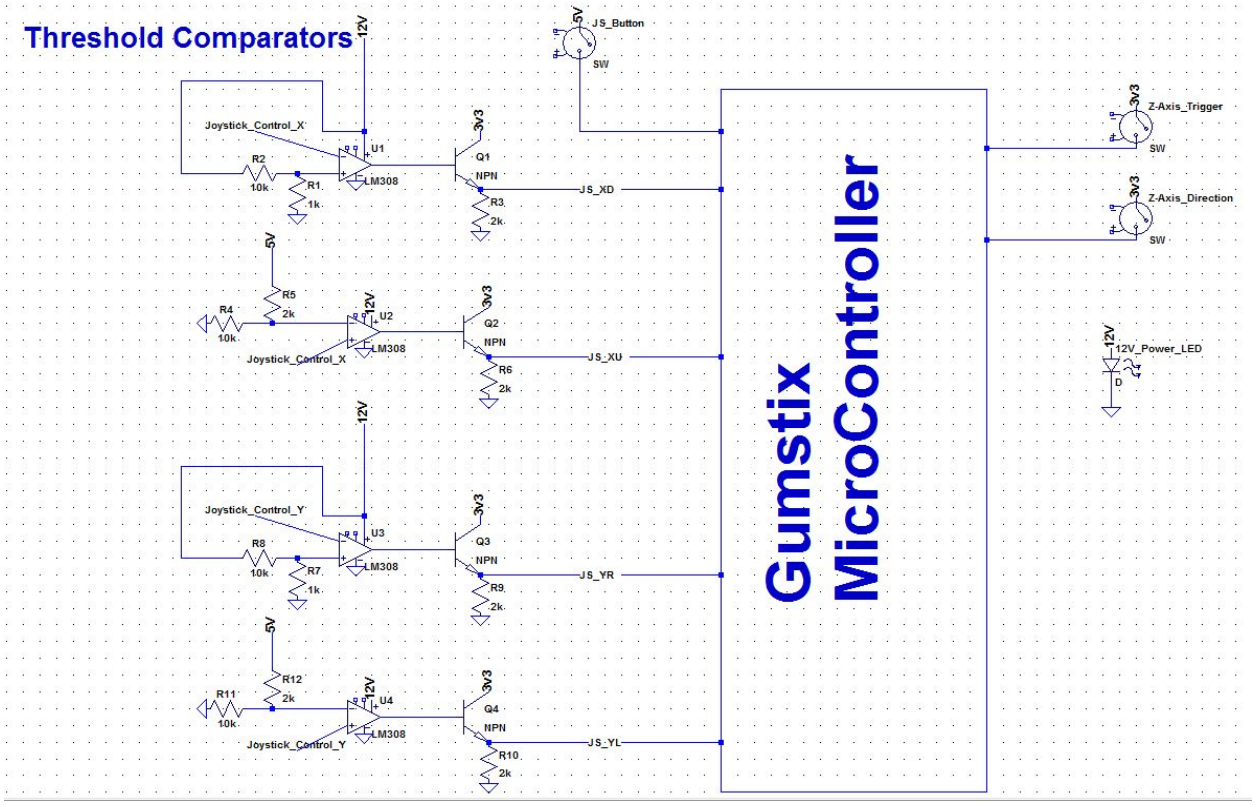

Figure 5 - Controller Circuit Diagram

d. Gumstix Controller Driver

The Gumstix Controller Driver utilizes the following files:

- gloveDriver.c
- Makefile

The glove kernel driver is written to interface with general purpose input/output (GPIO) pins that connect to the external controls that command the arm. These external controls are are discrete inputs coming from either a push button or dual threshold comparators. A total of seven (7) GPIOs are required to account for all of the discrete inputs that control the arm. The discrete inputs that use the dual threshold comparators are used on the analog joystick for x axis up, x axis down, y axis left and y axis right using GPIO pins 28, 29, 17 and 101 respectively. The joystick button uses GPIO pin 30 which controls the open/close state of the gripper. The tilt sensor as defined in the code uses GPIO pin 31, which was replaced with a push button as well controls the wrist movement. This was replaced with a push button, as opposed to the tilt sensor, because the tilt sensor was not very accurate and was found to provide very erroneous data. Lastly, the final push

button associated with the controller uses GPIO pin 16 and controls the direction of the wrist movement.

The kernel driver uses the standard functions for a basic driver which includes open, release, read, write, exit and init. Open, release, read and write are defined in the file operations data structure used for kernel drivers. Open and release are not required to do anything and as such are empty functions that just return zero. They only exist because they are required for compilation. The write function as well is not required to do anything and only returns zero. The write function could have been used, given enough time, to initiate calibration for sensors as the original design called for.

The read function provides a means for the glove user level program to communicate with the driver and to get the commands from the driver to send to the arm. When reading the device file, it will simply print four numbers, corresponding to commands for the elbow, shoulder, wrist, and gripper in that order. Since all inputs are discrete, all the commands are simplistic. For the x axis, or elbow, a 0 means do nothing, 1 means move up, and 2 means move down. For the y axis, or shoulder, a 0 means do nothing, 1 means move left and 2 means move right. For the z axis, tilt (as defined in the code) or wrist, a 0 means do nothing, 1 means move forwards and 2 means move backwards. Lastly, for the gripper, a 0 means open the gripper and 1 means close the gripper.

During initialization of the kernel driver, it first registers the major number with the operating system with a value of 61. It then allocates space for a globally defined buffer to be used with the read function of the driver. It then requests use of the necessary GPIO pins and sets their directions as inputs. Lastly it sets up a globally defined timer and sets its expiration of the period defined to be 100 Hz (10 ms). 100 Hz was chosen as the sampling rate to give us real time control of the arm even though the user space only samples the commands at 10 Hz (100 ms). At the time of the demo, we were still optimizing the performance and it could still be improved upon.

The only other function defined in the kernel driver is the callback function which is called every time the kernel timer expires. This is the function where it actually samples all of the inputs and sets the commands accordingly. This function first samples all of the GPIOs at once right after the other and based on whether or not they are asserted, it will set the command as either 0, 1 or 2 as defined above based on their function. After setting up the commands, it resets the timer to assure continuous sampling of the GPIO pins.

e. Gumstix Controler User-Level Program

The Gumstix Controller User-Level Program utilizes the following files:

- gloveBluetooth.c

- gloveBluetooth.h
- main.c
- Makefile

The user space level program that runs on the Gumstix controller is the program that interfaces with the custom kernel driver and sends the commands to the arm over a bluetooth connection. The code for bluetooth resides in gloveBluetooth.c and gloveBluetooth.h. The file gloveBluetooth.h really only contains a definition for the MAC address for the Raspberry Pi's bluetooth adapter so that it can create a connection and a function prototype for the function that initializes the bluetooth connection. The file gloveBluetooth.c contains the implementation for the bluetooth initiation. This function opens a socket and attempts to connect to the Raspberry Pi as a bluetooth client. Once a connection has been established, it will return the file descriptor of the connection socket.

Once the bluetooth connection is made, the glove program then enters an infinite while loop that opens the '/dev/gloveDriver' file, reads the data from the driver and packs it into a 4 byte data structure. It then closes the 'dev/gloveDriver' file and writes the data structure to the bluetooth connection. Once it has completed this operation it sleeps for 100000 us, making the loop run at 10 Hz.

f. Raspberry Pi Data Router

The Raspberry Pi Data Router utilizes the following files:

- main.c
- Makefile

The Raspberry Pi acts as the communication hub between the Gumstix controller and the Arduino controlling the robotic arm. The source for this code is dependent on the BlueZ bluetooth library and needs to be installed on the Raspberry Pi. The source code itself is very simplistic.

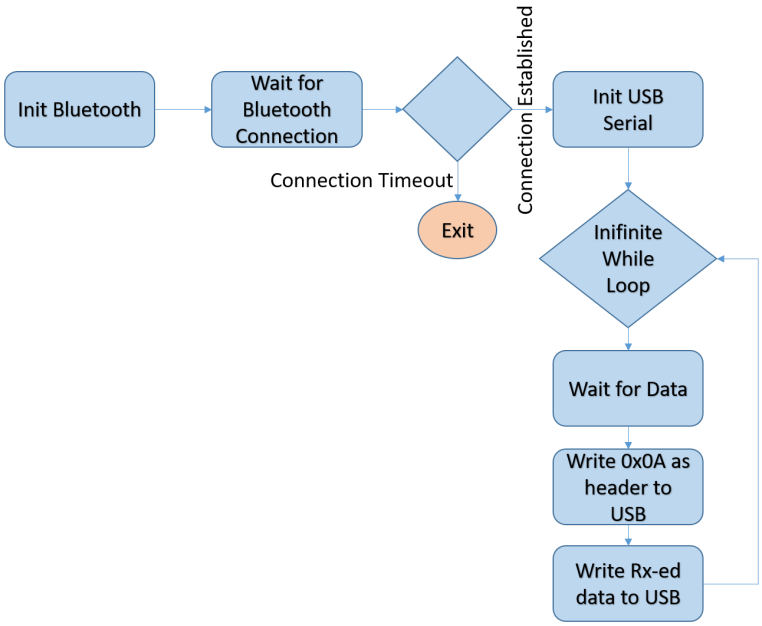

Figure 6 - Raspberry Pi Data Router Flow Diagram

As shown in Figure 6, the first thing that the data routing software does is initializes the bluetooth connection. It opens up a bluetooth socket and binds it to the Pi's address. It then listens for any incoming connections keeping in mind that the Raspberry Pi and Gumstix have been previously paired. Once a connection is established, it then proceeds to initialize the USB serial connection.

The USB serial connection uses the device '/dev/ttyACM0' as the connection and sets up the connection as read/write. After it opens the device, it configures the serial connection at 9600 baud and the other default options that the Arduino uses for its native serial connection.

Once both connections have been established, the program enters an infinite while loop that just waits for incoming data via the bluetooth connection made with the Gumstix controller. Once a packet of information has been received from the Gumstix, it then writes a byte of 0x0A as a header and then the received packet of information. The packet of command data received from the Gumstix controller consists of 4 bytes of information. The first byte contains the command for the x axis, or elbow, of the robotic arm. The second byte contains the command for the y axis, or shoulder, of the robotic arm. The third byte contains the command for the z axis, or wrist, of the robotic arm. The fourth and final byte contains the command for the gripper. Once the command data packet is written, it then listens to the serial port as the Arduino echos commands as acknowledgement. After this the loop returns and goes back to waiting for more data from the Gumstix controller.

g. Arduino Arm Controller

The Arduino Arm Controller utilizes the following files:

- Robotic_Arm.ino

The Arm Controller code, attached, initializes all of the motor controller pins including 4 PWM pins and 6 digital pins for all motors and servos. The setup() sets all of these pins as OUTPUTs, initializes the servo motor using Servo.h, and also initializes the serial port to a baud of 9600.

Arduino code runs in an infinite loop once the board has completed all setup. The loop function begins by waiting and listening to the Serial port until it reaches 5 bytes of data i.e. Header, X, Y, Z, Gripper. It stores all of this data into individual variables and proceeds to run down if statements for each degree of freedom. Depending on the values entered (usually either 0, 1 or 2) which indicates do nothing, move in a positive direction, or move in the negative direction. These values are standard for all of the different inputs.

Each motor movement possesses its own function which is called if the input data indicates the desired movement. Each function will change corresponding digital and PWM controls to the specific motor being triggered. The chosen speed for the motors is a value of 200 due to wanting to minimize this number to slow down the DC motors, any lower and the DC motors fail to operate properly. Both the wrist and elbow motors contain if statements that read in data from the potentiometers, however these limiting functions are not complete due to the complexity of dual variable mapping as a limiting function.

h. Arm Circuitry

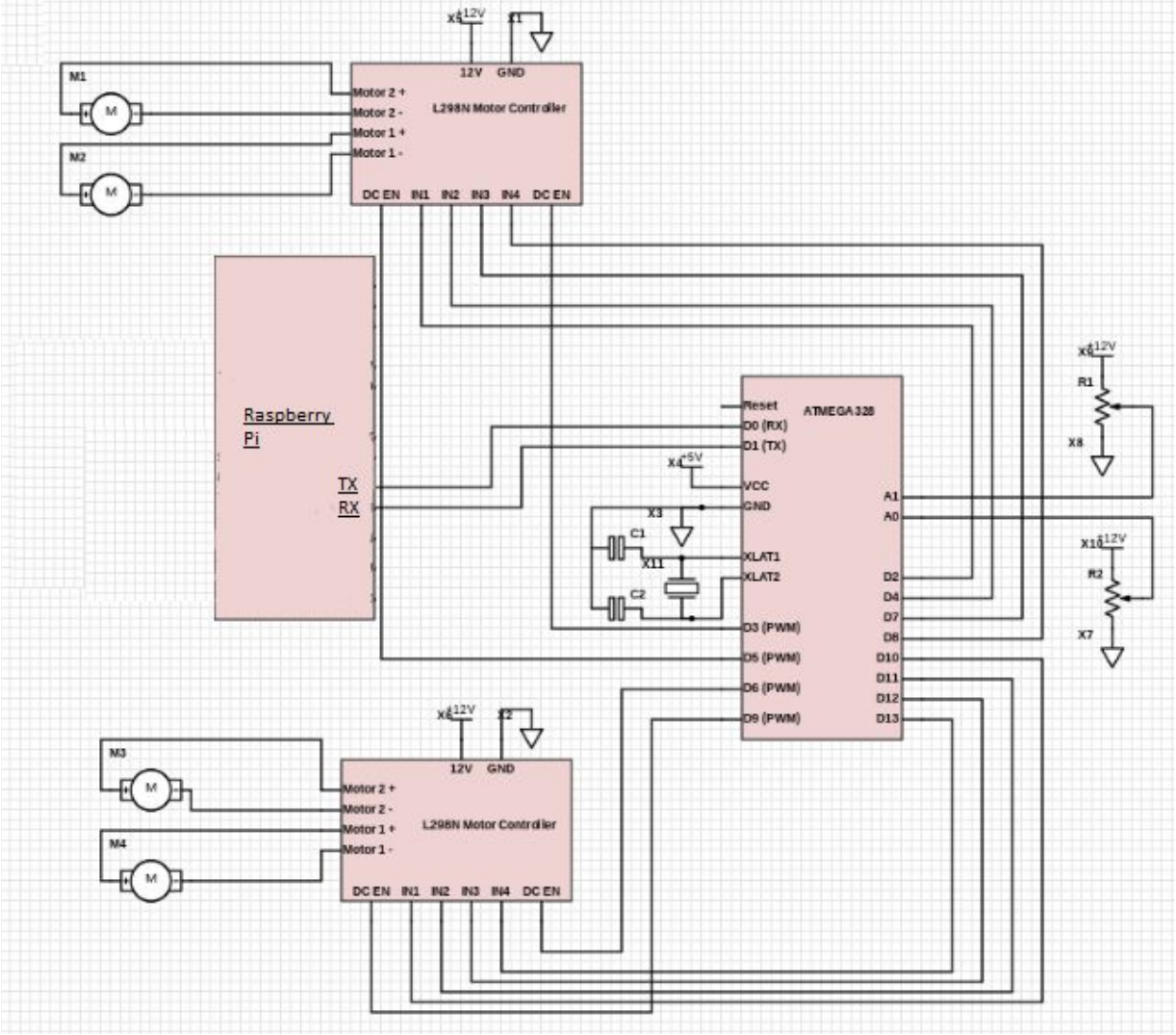

Figure 7 - Arm Circuit Diagram

4. Summary

The original goal of the project was to produce hardware and code to control wirelessly a robotic arm that could perform fine motor controls. We have implemented this by modifying an already designed robotic arm with 3 degrees of freedom with motor drivers and analog potentiometers. The controller utilizes various push buttons and an x-y joystick that is able to control all axis of the robotic arm including the gripper. While the live demonstration did not necessarily display the full capability of the robotic arm, in testing the arm was successfully able to pick up and move light objects to desired positions in the proximity of the robotic arm. The difficulty seen in the live demo shows the remaining challenges in the project including wire control, 2-D limit mapping (potentiometer data for wrist and elbow), and ease of use for the controller (button controlling Z-axis falling out).

References

[1] Marvell, “PXA270 Processor Datasheet”, Developers Manual, April 2009.

[2] http://www.instructables.com/id/Intro-and-what-youll-need/