

Clawbotics Control Center

Jessel Serrano, Nicholas Minckler

December 9, 2014

CEN 4065 Software Architecture & Design

Fall 2014

Instructor: Dr. Janusz Zalewski

Software Engineering Program

Florida Gulf Coast University

Ft. Myers, FL 33965

1. Introduction

This software design project is a continuation of a previous networking project in which a robot was remotely accessible and manipulated over the Internet from anywhere in the world [1]. The current project focuses on tracking the location of the robot before and after manipulation, allowing the user to know exactly *where* the robot is and how far it has traveled.

The previous project implemented their remote control via the VEX Clawbot (Figure 1), an instance of robotic kits offered by VEX Robotics [2]. Using the VEX Network Adapter on the Clawbot, remote and wireless control was achieved. A website was implemented that featured a panel to control the Clawbot via a network. Additionally, a network stream from the mounted IP camera was implemented to allow live video feed.

The current project will extend the web panel to include remote tracking. By installing additional sensors onto the Clawbot, such as the Integrated Encoder Module [3] as seen in Figure 2, the relative distance and angle to the starting location can be recorded with precision. These data are to be interpreted via client-side and then transformed into a graph to display the current position of the Clawbot as well as its starting position and path traveled.

The importance of these concepts and respective technologies cannot be understated. Remote accessibility, manipulation, and tracking are quintessential tools for missile guidance, deep-sea and space exploration, automobile navigation and counter-theft, etc. While this is a classroom project with student robotic systems, one cannot underestimate the striking similarity between the Clawbotics Control Center and the Mars Exploration Rover.

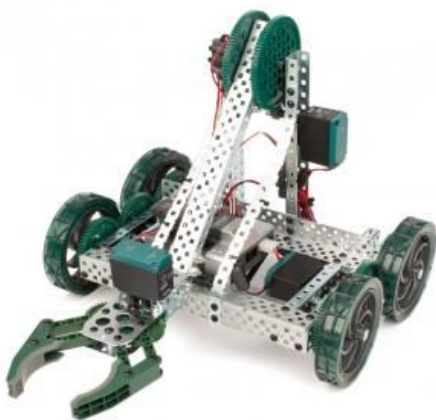


Figure 1: The VEX Clawbot



Figure 2: The Integrated Encoder Module

2. Definition of the Problem

2.1 Previous Project Description

With the development of more advanced and autonomous robotic systems, the need for software to control these machines is becoming more prevalent. First and foremost, the goal of this sort of software is to establish a form of control over a physical system, possibly remotely. The original project for this system decided to use the VEX Clawbot in conjunction with its wireless capabilities and a web-based control panel (Figure 3). This task was accomplished by a basic control system accessed through a website which allowed minimal controls and a utility to upload new software to the Clawbot. The finished design is a web panel, accessible via *vex.cs.fgcu.edu* (Figure 4).

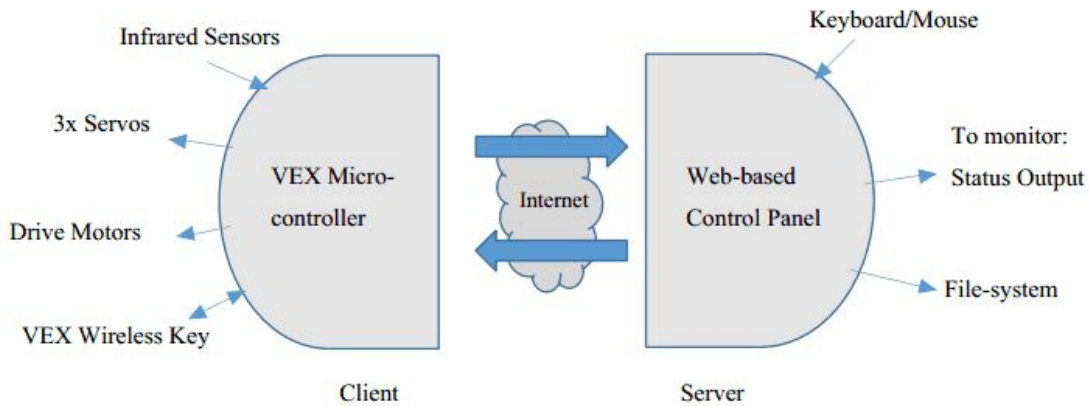


Figure 3: Original VEX ClawBot Context Diagram

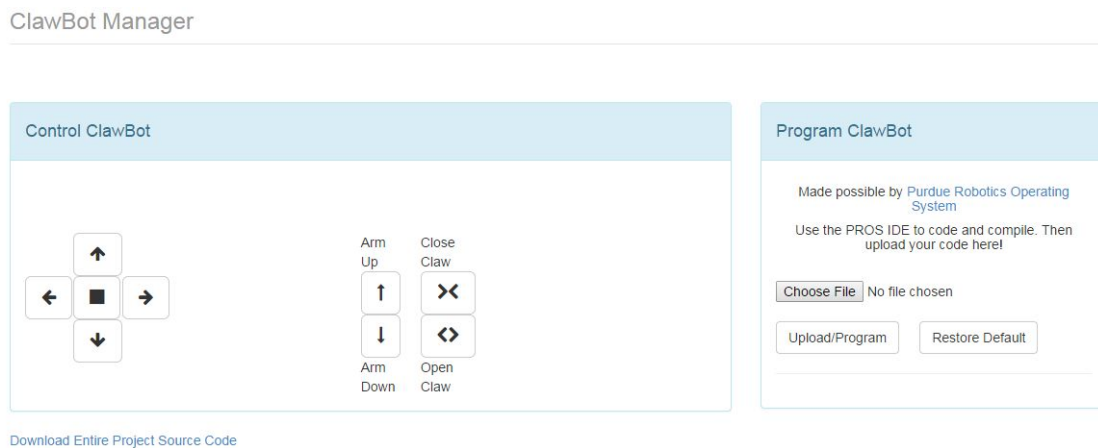


Figure 4: Original VEX ClawBot Manager

2.2 Goals of the Current Project

The current project intends to expand on this control panel, making use of the utilities already given, to create a more advanced method of controlling and tracking the Clawbot. At present, there are only a few commands available, such as “Go, turn, stop. Lift/drop and open/close claw.” The purpose of this project is to allow a user to track the location and movement of the Clawbot via the web-based control panel by utilizing the sensors attached to the physical system. In addition, the implementation of a control scheme that allows the user to map a route for the Clawbot to follow is a further problem that, if implemented, would allow a two-way interaction with the Clawbot.

The goals of the project are illustrated on the following diagrams. Figure 5, which represents the Physical Diagram, shows the physical configuration of all entities involved.

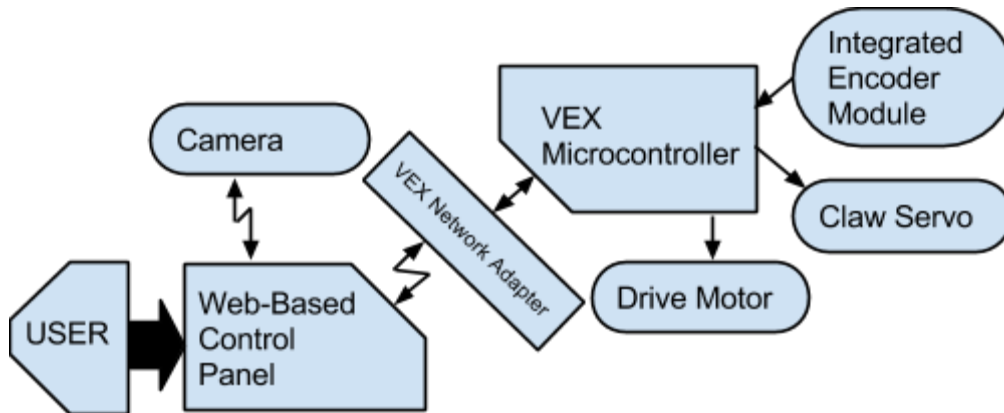


Figure 5: Current ClawBot Physical Diagram

A brief description of functions of all devices is given below:

- The VEX Microcontroller must be turned on before operation.
- The user logs in to the control panel and is presented with a visual display consisting of a video feed and control scheme.
- The camera, which is physically attached to the VEX robot chassis, wirelessly transmits the video feed to the control panel. The camera operates apart from the VEX Microcontroller and sends its data to the Web-Based Control Panel through its own hardware and wireless capabilities.

- Any controls the user inputs to the control panel are transferred wirelessly to the VEX Network Adapter, which then transmits the instructions to the microcontroller.
- The microcontroller transmits instructions and voltage control to the motors and servo to move the robot.
- The Integrated Motor Encoder (known as the IME), attached to the robot's 'VEX 2-wire motor 393' drive motors, measures how fast the robot is going, what direction it is traveling in, and how far it has traveled. It sends this data to the microcontroller, which uses the VEX Network Adapter to transmit data to the Web Control Panel. The physical device is known as an Integrated Motor Encoder, while the respective software/firmware and functionality is known as the Integrated Encoder Module.
- The Web Control Panel uses the feedback data from the IME to create new instructions for the robot to follow and uses the VEX Network Adapter to transmit this data.

The context diagram for the software is shown in Figure 6. In this diagram, the Clawbotics Control Center is shown as a combination of both a Web-based Control Panel and the VEX Clawbot Microcontroller. These two components communicate over a network. Each component has multiple devices attached. These devices send and receive signals to and from either the control panel or the microcontroller. The devices, used by the Web-based Control Panel, are the keyboard and mouse, the visual display linked to the computer and a wireless camera that is physically attached to the Clawbot. The devices attached to the VEX microcontroller include two IMEs (one for each motor), two drive motors and a claw servo.

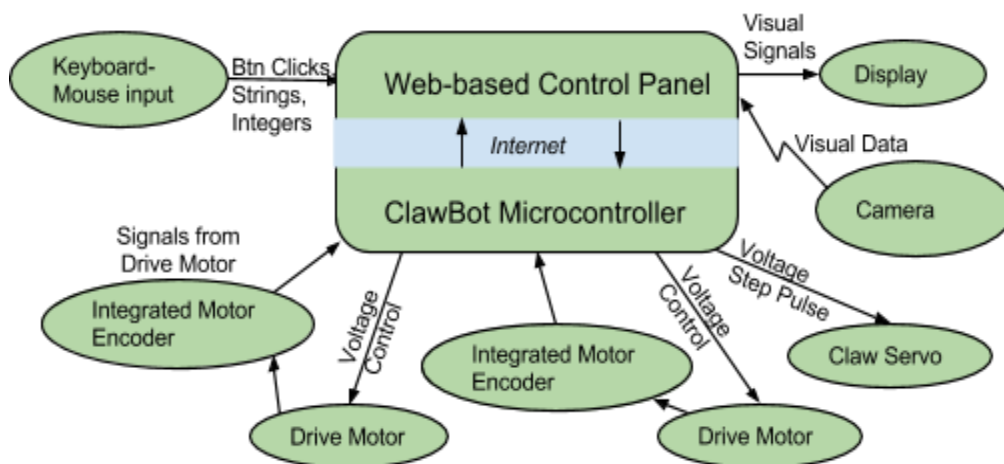


Figure 6: Current ClawBot Context Diagram

2.3 Software Requirements

In addition to the functionality described in previous sections, a use case diagram in Figure 7 illustrates the robot operation from a user's standpoint.

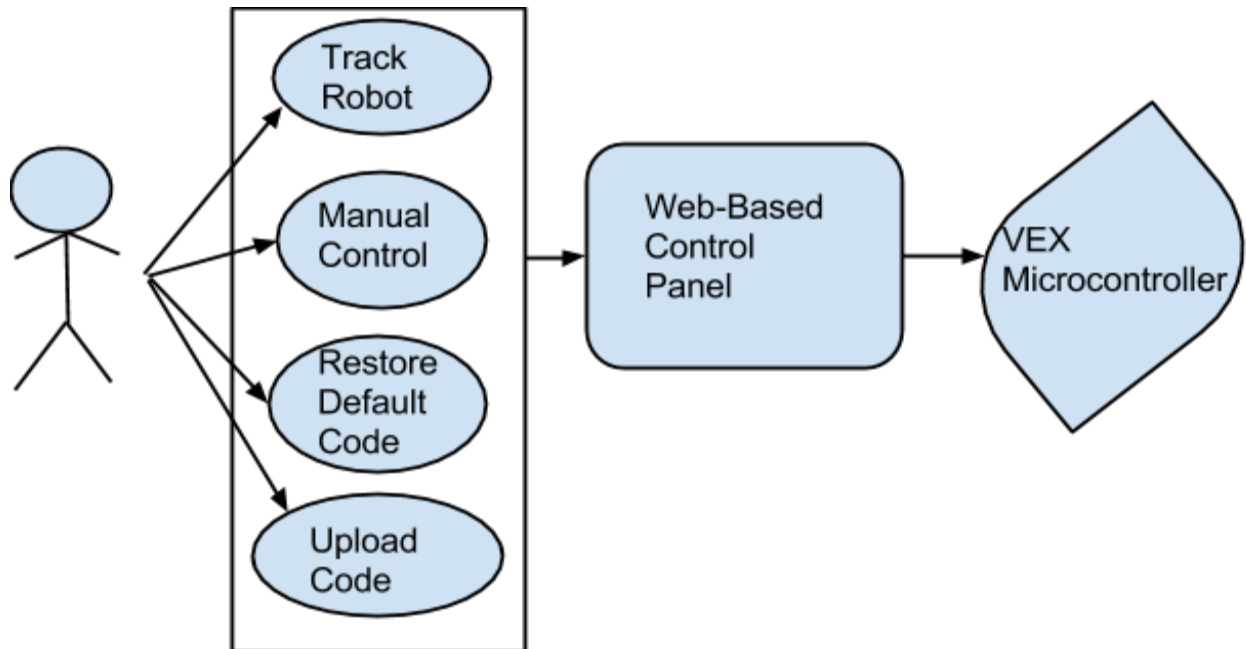


Figure 7: Use Case of C3 Software

2.3.1) Clawbot Microcontroller Software (CMS)

2.3.1.1) Input Requirements

2.3.1.1a) The CMS shall take signals from the Integrated Motor Encoders.

2.3.1.1b) The CMS shall receive instructions from the WCP through the network adapter module.

2.3.1.2) Output Requirements

2.3.1.2a) The CMS shall send On/Off signals to the drive motors to control their rotation.

2.3.1.2b) The CMS shall send step signals to the claw servo to open and close the claw.

2.3.1.3) Functional Requirements

2.3.1.3a) The CMS shall use the data sent by the WCP to control the drive motors and claw servo.

2.3.1.3b) The CMS shall use the data sent by the WCP to reprogram itself.

2.3.1.3b) The CMS shall use the signals received by the Integrated Motor Encoders to determine its direction, distance and speed.

2.3.2 Web-based Control Panel (WCP)

2.3.2.1) Input Requirements

2.3.2.1a) The WCP shall take keyboard/mouse inputs in the form of button clicks, strings and integers.

2.3.2.1b) The WCP shall take visual data from the camera attached to the robot's chassis.

2.3.2.2) Output Requirements

2.3.2.2c) The WCP shall send data (navigational signals and code) to the CMS through a network.

2.3.2.2d) The WCP shall send visual signals to a display.

2.3.2.3) Functional Requirements

2.3.2.3a) The WCP shall be displayed to the user in a web browser.

2.3.2.3b) The WCP shall allow the user to control the drive motors and claw servo on the VEX Clawbot through a network in a web browser.

2.3.2.3c) The WCP shall display a live video feed using data from the camera in a web browser.

2.3.2.3d) The WCP shall display a Cartesian graph depicting the location of the VEX Clawbot and the path it has traveled in a web browser.

2.3.2.3e) The WCP shall allow the user to upload his/her own code to the VEX Clawbot Microcontroller over a network.

2.3.3 Non-Functional Requirements

2.3.3.1) Safety Requirements

2.3.3.1a) The CMS shall set the motor speeds to reasonable limits so as not to harm the environment or the Clawbot.

2.3.3.2) Security Requirements

2.3.3.2a) The WCP shall restrict access to the Clawbot controls to users who do

not enter the correct login and password.

2.3.3.2b) The WCP shall communicate with the CMS through a dedicated serial port.

3. Design Description

3.1 Architectural Design

The ClawBot Control Center (C3) software is composed of two parts: the Web-Based Control Panel (WCP) and the VEX ClawBot Microcontroller Software (CMS) (Figure 9). The WCP is the processing portion of the system, receiving data sent from the microcontroller and turning it into both a visual display and additional instructions for the VEX ClawBot to follow. The CMS, located on the ClawBot itself receives data from sensors and uses instructions sent from the WCP to command motors and servos.

The architecture design of the WCP is shown in Figure 8:

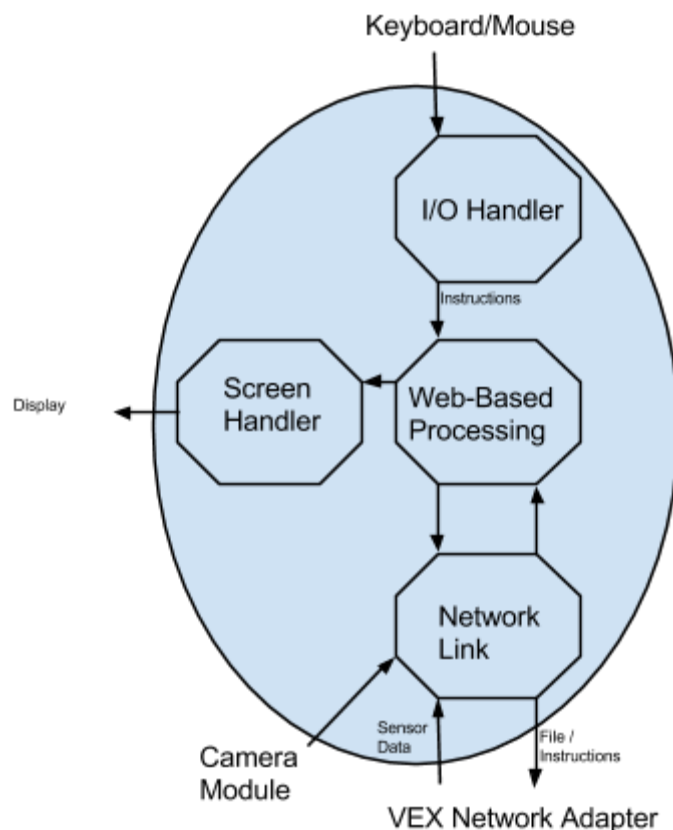


Figure 8: Architectural Diagram of Web-Based Control Panel

The WCP exists as a server on the Internet and is composed of:

- A Screen Handler to output visual data and create an IO system
- An I/O Handler to accept those key presses
- A processing unit to turn data received into usable instructions for the ClawBot software.

- A Network Link to the Internet to both send those instructions to and receive sensor data from the ClawBot.

The Web-based Control Panel combines these four components to create a coherent interface for the system.

The CMS's architecture is shown in Figure 9.

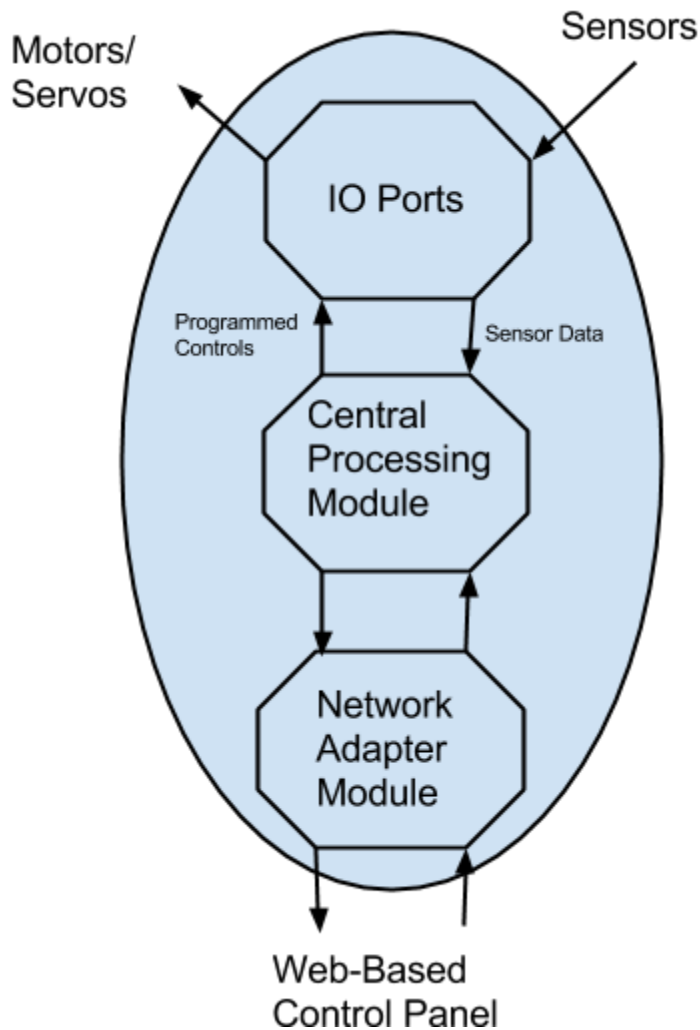


Figure 9: The Architectural Design of the VEX Clawbot Microcontroller Software

The microcontroller is the hardware portion of the system which the WCP communicates to via the VEX Network Adapter attached to the ClawBot to allow wireless communication to the Internet. The microcontroller has multiple I/O ports for external devices. The CMS can handle sensors and motors attached via the I/O ports with an I/O Port Module. The data from sensors and instructions for the motors are transmitted to and from the WCP respectively, via the

Network Adapter Module. The operation of these two modules is coordinated by the Central Processing Module.

3.2 Detailed Design

Since this software has two main parts, the control panel (WCP) and the microcontroller software (CMS), detailed designs are needed to describe the structure of each part. Figure 10 is a simple structure chart that demonstrates the dynamic structure of the WCP, while Figure 11 displays the dynamic structure of the CMS. Both diagrams represent a detailed view of the previous two figures. Each module is expanded to reveal the dynamic interaction of certain software. The modules shown in Figure 10 and 11 that do not reveal inner modules are design elements that are pre-written or hard-coded. For example, Figure 10 displays the I/O Handler as a single module because what handles the input signals from the keyboard and mouse is the operating system of the computer being used (Windows, Linux, etc.); whereas the Screen Handler is a module with more detail due to the fact that what is being presented is the software being designed.

Following this, several more modules can be seen in Figure 10, including the tracking module (the main extension of this project), the upload-code-to-robot module, and the video streaming component. All these modules are contained within the Screen Handler module, which essentially, builds the web page for users. The web page module is a higher class that contains the other three modules. All of these modules both send and receive data through the Network Adapter module, which is all mediated by the Web Based Processing module. This data is transformed to a visual representation of the robot's controls and current location and displayed using the I/O Handler, a hard-coded module found in Figure 10.

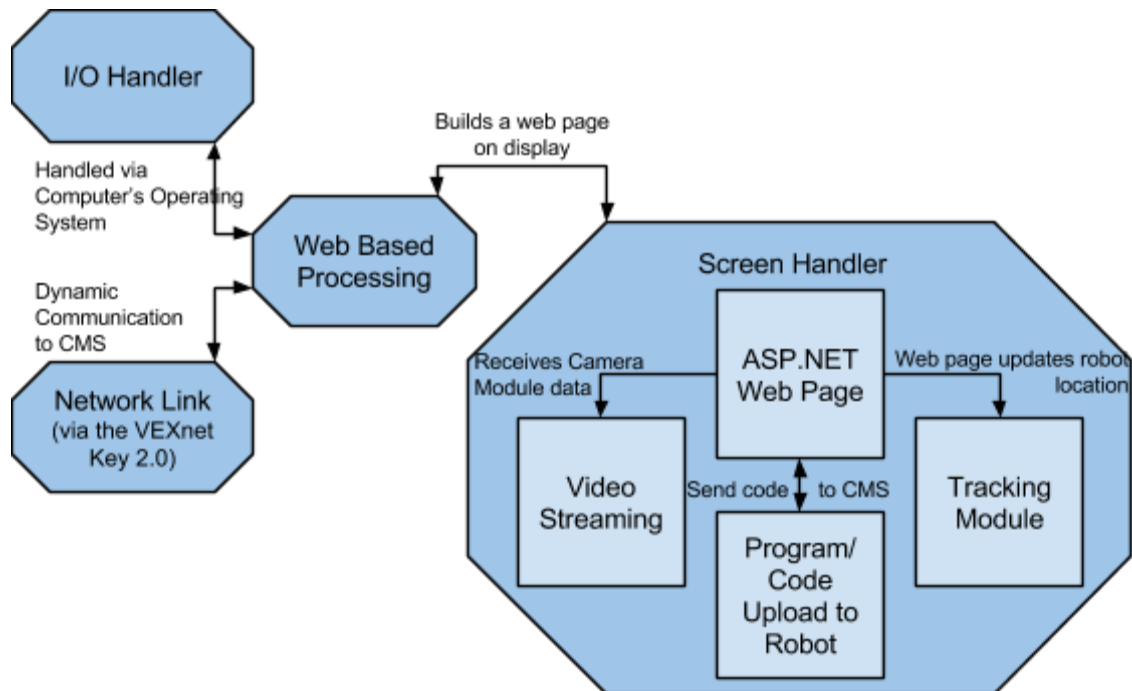


Figure 10: Structure Chart of the Web-based Control Panel

The same pattern can be seen from Figure 9 to Figure 11. In Figure 9, the Architectural Diagram of the Clawbot Microcontroller Software can be seen from a static view. Figure 11 expands this diagram to reveal the dynamic modules of each module in Figure 9. For example, Figure 11 contains several modules that operate in the Central Processing Module. These include the Integrated Encoder Module, the Navigation Module, and the Motor Control Module. These three modules interact dynamically to control the robot. The Integrated Encoder Module reads feedback signals from the motors. These signals represent integers that increments based on how much the motor has turned. The navigation component then receives these integers from the IMEs. These integers are then transferred from the microcontroller software to the Network Adapter Module, which is simply the firmware/functionality of the VEXNet Wireless Adapter/Joystick combo; this is why this module is self-contained: all the code is pre-written and pre-packaged. Once the data is sent across the network, it will be transformed via the WCP.

Concurrently, the CMS will take in data from the WCP through the Network Adapter Module. This data will be commands to the motors of the Clawbot, which is handled by the Motor Control module. The motors are actuated via signals on the microcontroller of the

Clawbot. These signals are both read and written via I2C protocols, which is a daisy-chaining protocol embedded on the microcontroller.

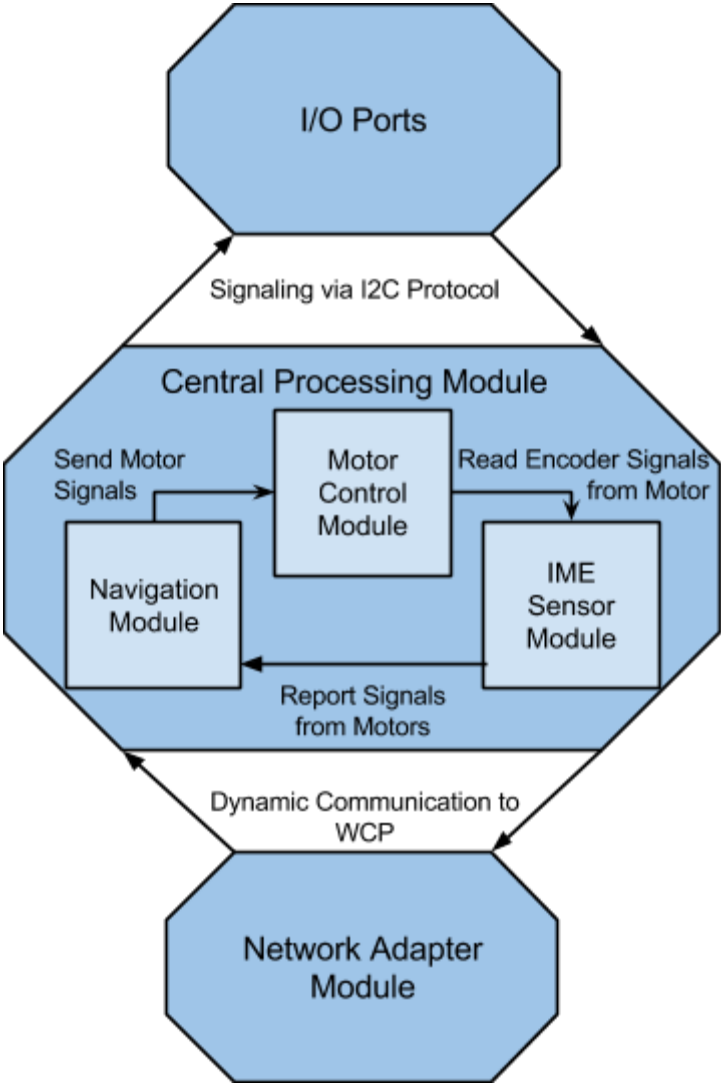
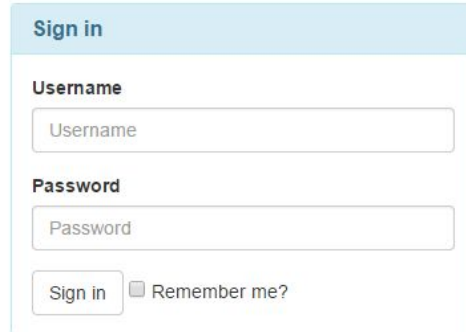


Figure 11: Structure Chart of the Clawbot Microcontroller Software

3.3 Graphical User Interface

When the user goes to the WCP at `vex.cs.fgcu.edu`, they are presented with a log in prompt as shown in Figure 12. The site is created through the ASP.NET framework.



The image shows a 'Sign in' form with a light blue header. Below the header, there are two input fields: 'Username' and 'Password'. Below the 'Password' field, there is a 'Sign in' button and a 'Remember me?' checkbox.

Figure 12: WCP Login Prompt

When logged in, the user is presented with the control and tracking screen, as shown in Figure 13. There are forward, back, turn left, turn right, and stop controls for the drive motors, and arm up, arm down, claw open, and claw close controls for the claw arm of the ClawBot. Besides the direct controls for the ClawBot, there exists a programming utility that allows the user to upload new code or restore the default settings to the ClawBot.



The image shows two side-by-side panels. The left panel, titled 'Control ClawBot', contains a directional pad with four arrows (up, down, left, right) and a central square button. To the right of the directional pad are four buttons: 'Arm Up' (up arrow), 'Close Claw' (two crossed arrows), 'Arm Down' (down arrow), and 'Open Claw' (two arrows pointing outwards). The right panel, titled 'Program ClawBot', contains text: 'Made possible by Purdue Robotics Operating System', 'Use the PROS IDE to code and compile. Then upload your code here!', a 'Choose File' button with 'No file chosen' text, and two buttons: 'Upload/Program' and 'Restore Default'.

Figure 13: WCP Controls

4. Implementation and Testing

The implementation of the Clawbotics Control Center requires the implementation of two pieces of software: the implementation of the Web-based Control Panel (WCP) and the implementation of the Clawbot Microcontroller Software (CMS). The previous report [1] had already implemented a basic software on the Clawbot microcontroller that allows the:

- Forward and backward movement of the robot
- Left and right turning of the robot
- Arm up and down of the robot's claw arm
- Claw close and open

4.1 Implementation of Clawbot Microcontroller Software

The VEX ClawBot has most of its functionality embedded in the firmware of the microcontroller that cannot be touched. The software which controls the robots movements by sending and receiving data is created via the use of a provided API by the Purdue Robotics Operating System (PROS). This API is utilized using PROS' IDE, which is an Eclipse extension that incorporates PROS' libraries for the VEX.

Since this project is concerned with implementing a tracking component, the previous software on the microcontroller will remain generally untouched. The main method used by the VEX microcontroller (also called the VEX Cortex) is `void operatorcontrol()`. This method defines all controls that are not hard-coded into the device or initialized on start up. The modules displayed in Figure 11 are defined here.

The Navigation Module is comprised of two commands, `getchar()`, and a switch statement. The former reads a single character off of the terminal sent from the WCP and returns it as an integer. That integer is then sent through the switch statement to trigger the motor control.

```
// This sample code moves the robot forward at speed 75
while (1) {
    int command = getchar();
    int leftEncoder;
    int rightEncoder;
    char * encoders;
    switch (command)
```

```

{
    case 119: //if command is "w", go forward
        motorSet(10, 75);
        motorSet(1, 75);
        motorSet(6, 0);
        motorSet(7, 0);
        break;
    ...
    case 102:
        imeGet(0, &leftEncoder);
        imeGet(1, &rightEncoder);
        snprintf(encoders, 33, "%d,%d\n", leftEncoder,
rightEncoder);
        puts(encoders);
    ...
    case 103:
        imeReset(0);
        imeReset(1);
}
}

```

The Motor Control Module consist of multiple cases in the switch statement as triggered by the navigation module. When a specific character is sent, ‘w’ for example, the system sets all four motors of the robot to their respective values. The method it uses is `motorSet(int, int)`. The first argument, channel, ranges from 1-10 and dictates which motor is triggered. The second argument, speed, determines the speed of the motor and ranges from -127 to 127, where a negative number indicates reversing.

The hardware of the motors is physically connected to the integrated motor encoders, and whenever the motor rotates, the encoder reads these rotations. This data is stored on the encoders until the API is used to call upon the IME sensor module: `imeGet(int, int*)` retrieves the current value of the encoder specified by the first argument and writes it to the memory location specified in the second argument. Once it has this value, it uses `snprintf(char*, int, string, ...)` to format a string pointed at by the first argument, with size of the second, of format-string of the third, and values given to the format by all proceeding arguments:

```
snprintf(encoders, 33, "%d,%d\n", leftEncoder, rightEncoder);
```

The command used as an example writes a string to the pointer `encoders` of size 33. It is formatted to write as two doubles separated by a comma, in which the doubles are

`leftEncoder`, and `rightEncoder` previously received by using `imeGet()`. The WCP calls the VEX ClawBot to reset its values every time it writes, to ensure accurate readings. It does this by using the method `imeReset(int)`, which the ID of the encoder is the argument.

All three of these modules work in conjunction with one another. The encoder is physically built in to the motor and the navigation instructs both the sensors and the motors. In order to control the physical system, the API accesses the I/O ports using the I2C protocol, which is a daisy-chaining protocol that allows multiple physical devices to be daisy-chained, allowing one input socket on the microcontroller. The PROS API also uses pre-built functions to read and write from the WCP via the Network Adapter Module (which is composed of the VEXNet Key 2.0 and joystick combo).

4.2 Implementation of Web-based Control Panel

The WCP is a site constructed using the ASP.NET framework. It's primarily built using HTML, augmented with ASP.NET extensions and backed by a C# server-side code. The site compiles the code on start up and establishes its connection with the VEX ClawBot using a `SerialPort` as its Network Link.

```
SerialPort sp = new SerialPort(Session["vexPort"].ToString(), 115200,
Parity.None)
```

The `SerialPort` allows the server to communicate to the VEX joystick which then translates the instructions to the ClawBot. The Web-Based Processing is not called upon until the I/O handler receives a command from the user or a timer on the site's HTML:

```
<asp:button id="button" onClick="vexStop" />
<asp:timer id="timer" interval="500" onTick="read" />
```

Behind the HTML, the I/O handler uses ASP.NET's event handlers to catch the command and have the server process the request. Each event handler needs the object that sent it and the arguments said object sent.

```
protected void vexStop(object sender, EventArgs e)
{    writeCommand("z");    }
```

When the I/O handler has successfully received the instructions, the processing module takes over. The main method used is `void writeCommand(string command)`; The argument it receives is a single character that is determined based on command it received from the I/O handler, which it then uses to send the VEX ClawBot a command.

To send the command, it opens the port, writes the command, closes the port. Given the function succeeds, it will hide the page element named `lblOutOfRange`, or otherwise show it to alert the user the robot is out of range.

```
private void writeCommand(string command)
{
    try
    {
        sp.Open();
        sp.Write(command);
        sp.Close();
        lblOutOfRange.Visible = false;
    }
    catch (Exception)
    {
        lblOutOfRange.Visible = true;
    }
}
```

The details of each command are defined on the Clawbot itself, and the WCP gives up control once the command is sent. The commands currently accepted by the ClawBot are:

- Movement Commands (Forward, Backward, Left, Right, Stop)
- Claw Control (Claw Up, Claw Down, Claw Open, Claw Close)
- Integrated Motor Encoder Controls (Shutdown IMEs, Start IMEs, Get IME Values, Reset IME Values)

Each of these commands are assigned a character to be sent using the `writeCommand(...)` function.

The second portion of the processing module is the `read()` function. The timer on the ASP.NET site calls this C# function which writes, reads, then writes again to the CMS before converting the data received to a usable form.

```
private void read() {
    string temp = "";
    try
```

```

    {
        sp.open();
        sp.write('f');
        temp = sp.readLine();
        sp.write('g');
    }
    catch (Exception)
    {
        lblOutOfRange.Visible = true;
    }
    finally
    {
        sp.close()
    }
    encoders[] = temp.split({' ',' '});
    distance.push(new Tuple<encoder[0],encoder[1]>);
}

```

The port is opened to `sp.write('f')` to the ClawBot. 'f' is defined to write the encoder values to the server, which `sp.readLine()` then receives. `sp.write('g')` is then used to tell the ClawBot to reset the values after they have been read. in the `finally` statement, the port is closed to avoid errors. When the data has been received, the encoder values are split into two strings, one on each side of the comma. A list `distance<Tuple<int,int>>` stores the values of encoders in pairs to be used by the tracker module.

The last portion of the processing module interacts directly with the tracker module. While the tracker draws the data, the `calculate()` method takes values from `distance` to instruct what to draw. The implementation of this is described in detail in section 4.2.1.

The site also contains two additional features that were developed in the previous project: the first is the Video Streaming Module that is initialized using the following line in ASP:

```



```

The other is the ability to upload and restore the VEX ClawBot code to its default settings, or upload custom user-code. It accomplishes this by utilizing code created by the previous project to launch a .bat file located on the server.

4.2.1 Tracking Module

The module that receives the most attention for the purposes of this project is the Tracking Module, which tracks the location of the robot. This is implemented via the `TrackNMap` class, coded in C# and implemented on the back-end server side. The basic outline of the Tracking Module occurs in the following steps:

1. A new `TrackNMap` object is initialized in the `Page_Load` event of the ASP webpage
2. This object creates a new Cartesian plane in its constructor, which is placed on the Default web page
3. Data are then read from the CMS (IME values) and fed into the `TrackNMap` object
4. The `TrackNMap` object transforms the data into coordinates which represent the location(s) of the Clawbot
5. The `TrackNMap` object then updates and re-renders the Cartesian plane to show these locations and will draw lines to represent the path traveled

These five steps map onto requirement 2.3.2.3d, which states that “The WCP shall display a Cartesian graph depicting the location of the VEX Clawbot and the path it has traveled in a web browser.” In the following paragraphs, the details of each step will be expanded on using select code from the `TrackNMap` class.

The first step, creating the `TrackNMap` object, is done using the following line:

```
TrackNMap updateMap = new TrackNMap();
```

Once the object is initialized, a `Bitmap` object and a `Graphics` object are created to draw the basic Cartesian plane. This is done by drawing a rectangle and then each individual x-y line. This second step is implemented via the following code:

```
objBitmap = new Bitmap(400, 400);  
objGraphics = Graphics.FromImage(objBitmap);  
// Draw border  
Pen pen = new Pen(Color.Black, 3.0F);  
objGraphics.DrawRectangle(pen, 0, 0, 400, 400);  
// Draw x-y coordinate lines  
pen.Color = Color.Gray;  
pen.Width = 2.0F;  
objGraphics.DrawLine(pen, new Point(0, 200), new Point(400, 200));  
objGraphics.DrawLine(pen, new Point(200, 0), new Point(200, 400));  
// Draw rest of lines  
pen.Width = 1.0F;
```

```

for (int i = 20; i < 401; i = i + 20){
    objGraphics.DrawLine(pen, new Point(i, 0), new Point(i, 400));
    objGraphics.DrawLine(pen, new Point(0, i), new Point(400, i));
}

```

Once the Cartesian plane is reading, data can be read into the TrackNMap object. The data are read as ArrayList data structures, which contain lists of Tuple<int, int> objects. Much of the detail on how the data are read can be found above under the read() function.

The fourth step encapsulates most of the mathematics of this project, as it details the calculation and transformation of raw IME values to coordinates. Since the IME values are integers describing how much a particular motor has moved, direction needs to be abstracted from this data and transformed properly to display turning. This requires some constants on the Clawbot's side so as to properly determine the ratio of IME ticks to arc length (or degree of rotation). Through rigorous testing, the following constants can be approximated:

- One foot of Clawbot movement = 700 Integrated Motor Encoder ticks
- One 360 degree rotation of Clawbot = 1900 Integrated Motor Encoder ticks

The final step includes displaying the updated Cartesian plane on the website. This is implemented using ASP's Image Class, found under the System.Web.UI.WebControls namespace. The image is hosted on the web site using a imageUrl parameter, which loads a file path. The reason it loads a file path is because the TrackNMap object generates the Cartesian graph and all its plotted coordinates (Clawbot locations) as a JPEG image. This is demonstrated in the following code:

```

string mapPath = "~\cartGraph.jpg";
objBitmap.Save(mapPath, System.Drawing.Imaging.ImageFormat.Jpeg);

```

Once the graph is saved to the disk, the ASP code will load it from the disk onto the website. A sample of what the final graph will look like can be found in Figure 14. This figure displays the Cartesian graph along with plotted points and lines drawn from each point to represent the path the robot traveled.

Actual Result	Upon entering the correct credentials (fgcu; vex), the Default page that contains the Clawbot controls, the video feed, and the Cartesian Graph loaded successfully.
---------------	--

ID	2
Title	Motor Control
Related Requirements	2.3.1.1a, 2.3.1.1b, 2.3.1.2a, 2.3.1.3a, 2.3.2.1a
Directions	Upon logging in, attempt to use the controls presented. Press each control button and allow the robot enough time to respond.
Expected Result	Each button press should respond on the site based on browser styles of button clicks. If the clicks successfully are registered, the robot should respond accordingly based on the button pressed.
Actual Result	Success: Robot responds to commands, despite some lag at seemingly random intervals.

ID	3
Title	VEX Robot Programming
Related Requirements	2.3.1.3b, 2.3.2.3e
Directions	Upon logging in, press Upload Default.
Expected Result	If the uploading succeeds, the site will display a loading message and eventually display a success message, otherwise display an error.
Actual Result	Success: Site displays “Upload code” then “Successfully Uploaded!”

ID	4
Title	Integrated Encoder Data
Related Requirements	2.3.1.3b, 2.3.2.2c, 2.3.2.3d
Directions	Upon logging in and moving the robot, view the Cartesian graph.

Expected Result	When the robot is moved via its wheels, the Cartesian graph should display the movement of the robot.
Actual Result	TBD

ID	5
Title	Camera feed
Related Requirements	2.3.2.1b, 2.3.2.3c
Directions	Upon logging in, view the presented camera feed.
Expected Result	If the connection to the camera exists, the site should display a camera feed.
Actual Result	Success: the site displays a live feed of the camera when camera is turned on.

5. Conclusion

This project was centered around the VEX ClawBot which was programmed using C and controlled by a Web-Based Control Panel using ASP.NET and programmed in C#. The objective of the project was to establish a means to track the robot and display this tracking data in a meaningful way.

The previous project that had worked on the VEX ClawBot established a control panel and basic controls for the robot to control the motors attached to the physical system. Using the systems in place, this project built off of the existing infrastructure. Integrated Encoder Modules were installed on the motors to have a way to detect how far each wheels have moved.

The initial hurdle was finding a way for the ClawBot to communicate back towards the server. Due to the nature of serial ports and the security issues that could result when leaving such ports open, the reader/writer program had to maintain the serial port in such a way that it does not race itself and end up causing the port access to be denied altogether. When a reliable reading system was established, there was an issue with translating the data that was gathered into a meaningful display.

By observing the translation of encoder units to physical movement, an algorithm was developed to turn the forward-back measurements of each side of the ClawBot to a Cartesian graph that the user can view. Due to a physical limitation of the robot's motors, the mismatched strength of the hardware makes the tracking less than fully reliable. The tracking was established as best it could and serves to show a general route the ClawBot has taken.

6. References

- [1] M. Grojean, N. Hart, <Remote Vehicle Networking>, Florida Gulf Coast University, 24 April 2014, URL: <http://itech.fgcu.edu/faculty/zalewski/projects/files/VEXrobotics.pdf>
- [2] Guide for Building the Clawbot, VEX Robotics, URL: <http://content.vexrobotics.com/docs/instructions/276-2600-CLAWBOT-INST-0512.pdf>
- [3] Motor 393 Integrated Encoder Module, VEX Robotics, URL: <http://content.vexrobotics.com/docs/inventors-guide/276-1321-INST-0112.pdf>