5-2022

A Remote-Controllable Robotics Platform Based on the PIC16F88 Microcontroller

Clyde Johnson
Trinity University, cjohns14@trinity.edu

Matthew Roche
Trinity University, mroche@trinity.edu

Max Ulmer
Trinity University, mulmer@trinity.edu

A Remote-Controllable Robotics Platform Based on the PIC16F88 Microcontroller

Clyde Johnson, Matthew Roche, Max Ulmer

ENGR 4367-1

May 16, 2022

Pledged

Table of Contents

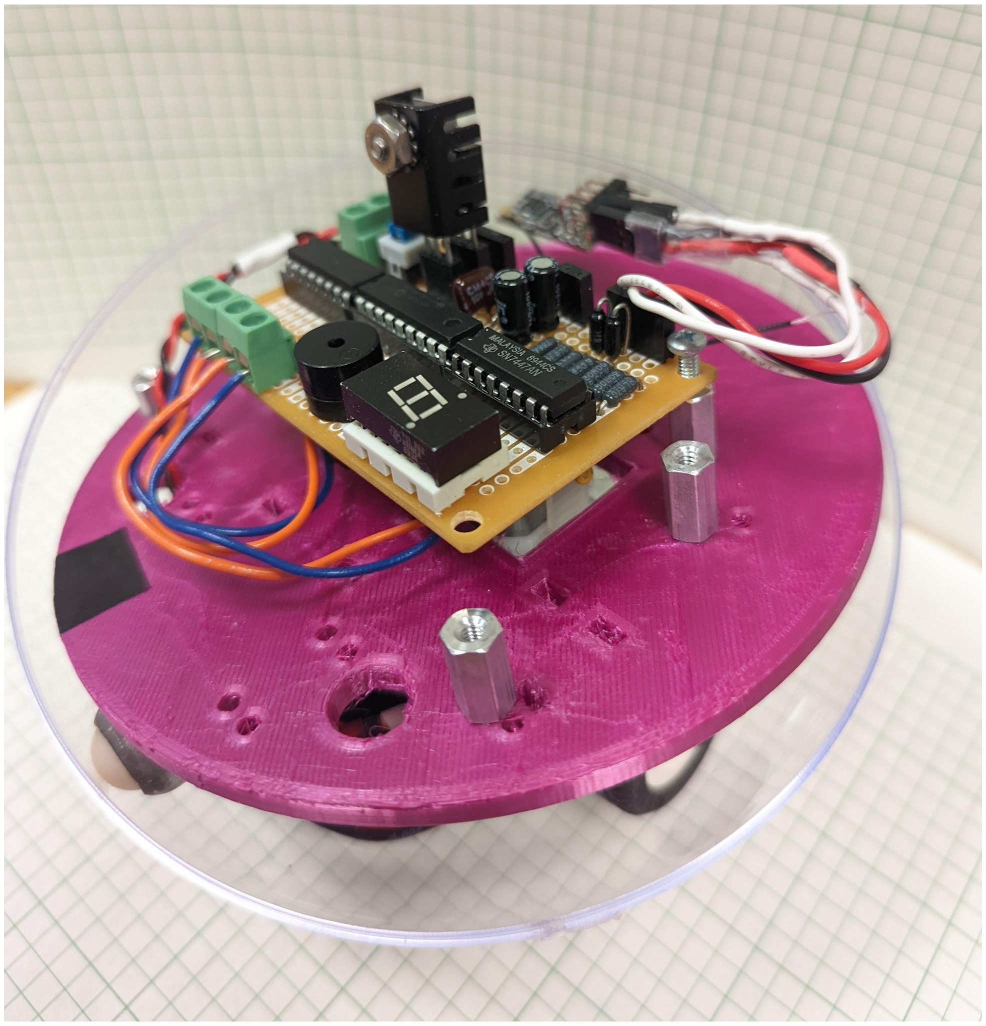
I. Design Summary

The goal of this mechatronics system design project is to create a simple robotics control and interface platform using a Microchip PIC processor. The intent is for this platform to be able to host a wide variety of simple sensors, displays, and outputs that can be carried around by its drive. The platform will be able to take in and interpret 2.4GHz remote-control signals.

For a novel twist on remote-control robotics, we have designed this robot to be encapsulated within an acrylic sphere. It is platformed on a 3D-printed chassis that holds upright its two geared DC motors that rest on the bottom of the sphere. At the midsection of the sphere sits the circular flat platform which carries the control board, sensors, and outputs. This setup can be seen in the final photograph of the project below. The robot's spherical design offers multiple mobility and maneuverability advantages over common four-wheeled chassis designs. It can traverse wet, sandy, or rough terrain without risking getting any individual wheel or axle mechanism stuck, and it is able to orient itself in any direction by revolving around a singular point. The shell allows the robot to remain intact in otherwise difficult environments, protecting its own electronics and sensor payload from being submerged in liquids or damaged by force.

For this project, we specifically implemented a battery level indicator and a tilt sensor to warn the user of instability or control movement. A common issue with less expensive RC vehicles is the sudden loss of power due to the battery dying; a battery sensor would eliminate this issue by displaying a number zero to five to indicate the amount of charge left. An orientation warning would make a loud beep when the interior structure of the robot is no longer the right way up, which causes issues when trying to steer. This sensor can be configured by the robot's processor to also control motion as a corrective aid. Additionally, the robot was designed to allow for easy customization once the base platform is completed.

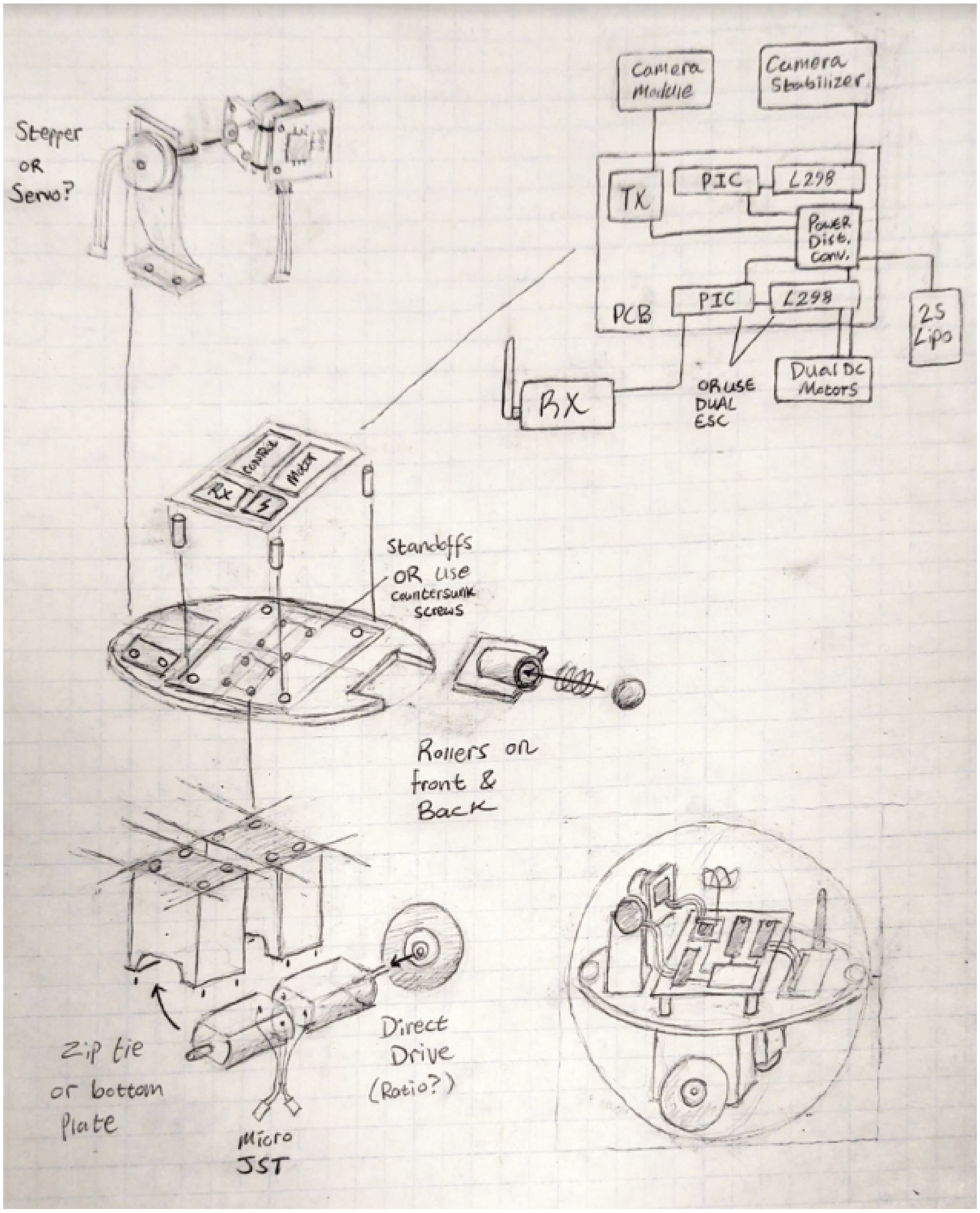
Photograph of Final Device



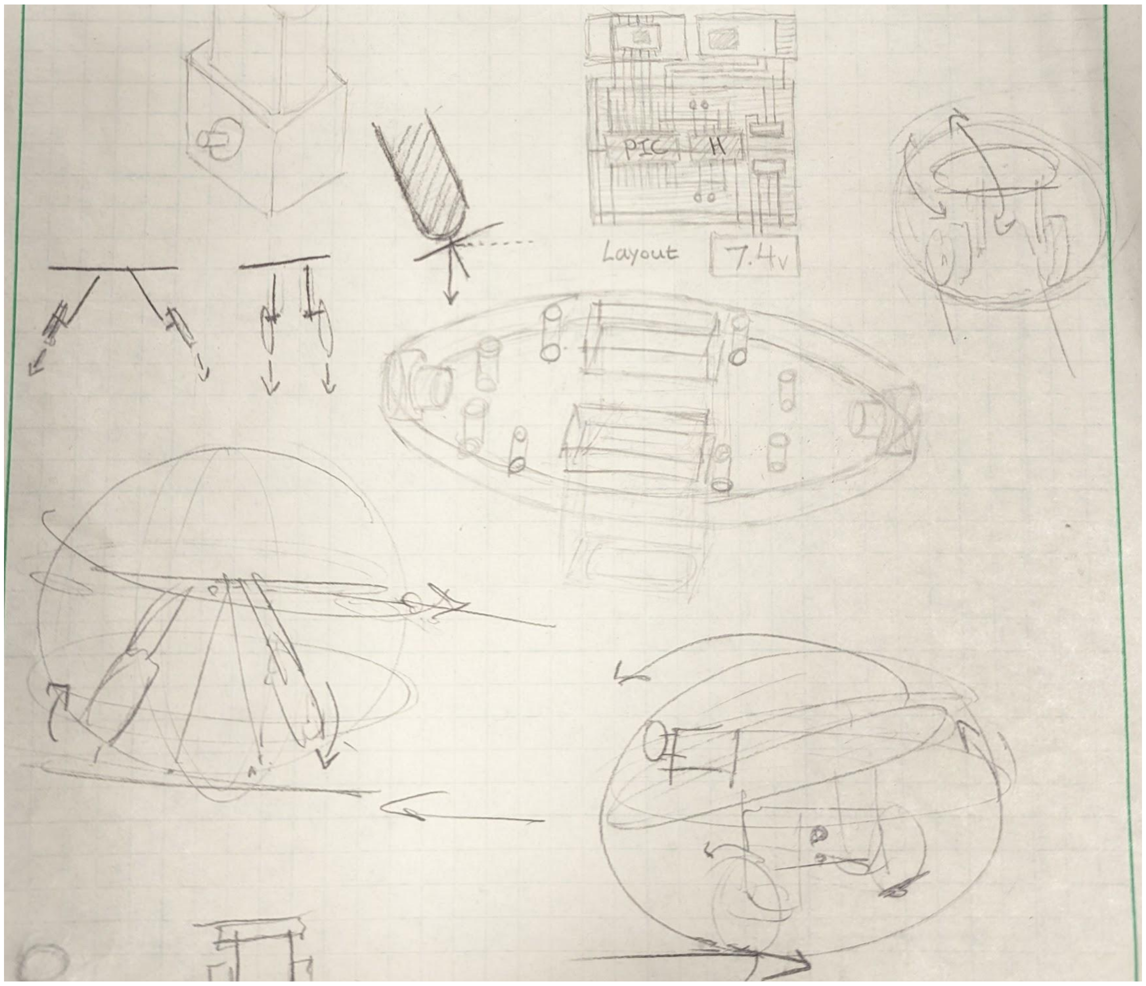
II. System Details

The initial concept for this device allows it to host any number of small sensors on its frame, contained within a clear acrylic shell. In our initial concept, a camera with a stabilization motor would be the main sensor used in our project. However, to ensure a working product would be completed on time, the camera was removed and replaced with a battery sensor and a tilt sensor. Below are sketches created during our early design process, planning out the basic circuitry required for this device, as well as brainstorming physical configurations for the board, sensors, and motors so that the chassis could be designed in Fusion 360 and produced to aid prototyping.

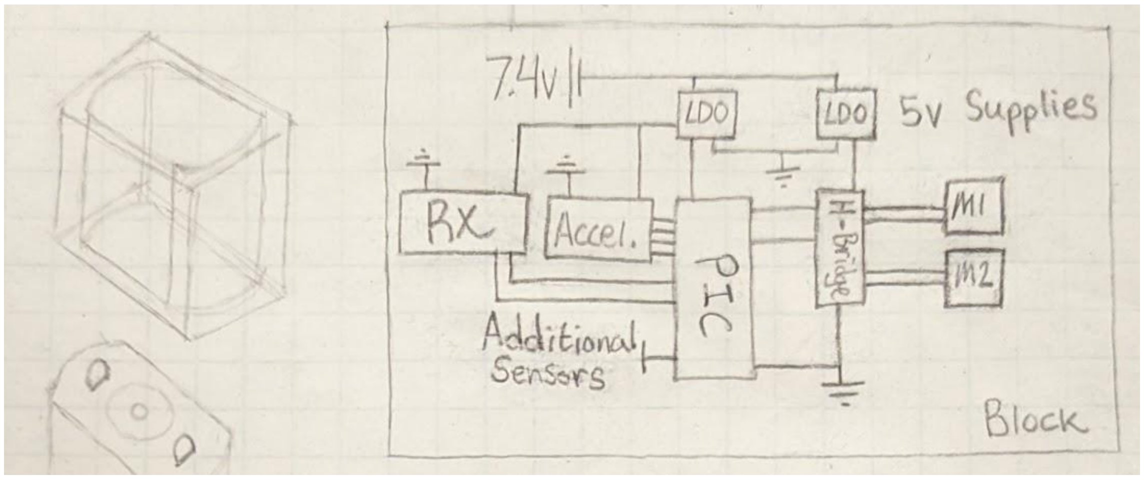
Initial Sketches of Basic Design and System Layout



Initial Sketches of Chassis Design



Refined Sketch of System Electronics Layout

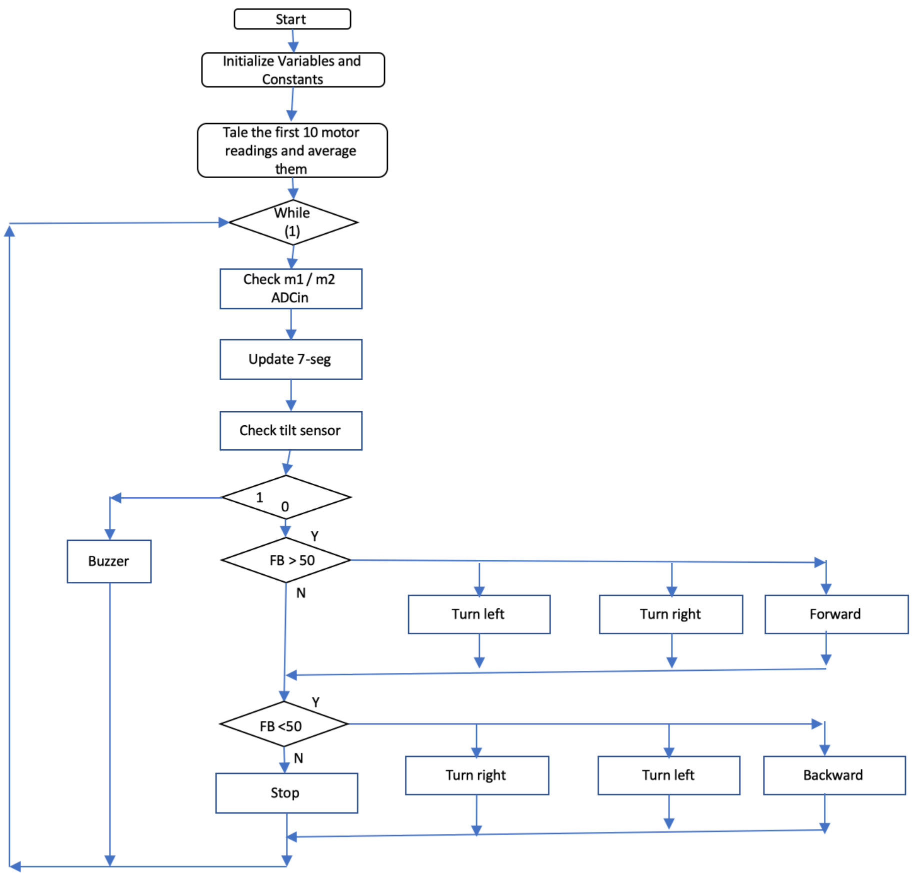


Once the specifics of how the PIC would be interfaced to its inputs and outputs were mostly nailed down in the initial design sketches above, work began on wiring up each individual subsystem in isolation on breadboards, and testing its functionality with basic programs on the PIC.

Some parts of the design were easy to interface - the warning buzzer and seven-segment ran off of 5V, and could be controlled using simple digital pulses. Wiring was straightforward. For the display, a SN7447AN BCD-to-seven-segment decoder chip was used to reduce the number of digital output pins required to operate it. By sending a four-bit number from four pins on the PIC, the seven segments could be lit in any configuration.

Driving the motors was similarly straightforward: The PIC read the incoming Left, Right, and Up, Down signals from the receiver on its analog to digital converter pins. These signals were converted to DC signals via passive lowpass RC filters. Then, the PIC averaged the signal values to mitigate noise from the filter. Next, the PIC outputs high or low to motor ports and pwm on a separate port to control the motor speed. The motors would be able to drive the robot in the four cardinal directions and the four corners. The result of this logic can be seen in the following figure.

Flowchart for the spherebot's programm



Things became much more challenging as we began to integrate all of these elements, and develop a coding scheme for driving the device using our RC transmitter and receiver pair.

III. Design Evaluation

The final prototype was operational in modular segments of the complete system. Functionality of the individual sensors was repeatable with marginal error in the amount of trials they failed versus the trials that were successes. We used a seven-segment display to show the battery level for our output display. The output from the PIC was a four bit number from zero to five. A BCD to seven segment display converter was then used to transform the signals from the pic to an output for the seven segment display. The PIC received a voltage from the center node in a voltage divider since the battery operated above the 5v maximum of a PIC port. This was an aspect that was successful. The use of the BCD to seven segment converter and

7-segment voltage polarity was not covered in the textbook. It did require some outside research, but not necessarily extensive. For this reason, we believe this section of the project deserves a 15.

For the audio output, we used a single tone buzzer. This fully satisfied the audio output requirement. However, this is the category that did require the least amount of additional research. This element was essential for debugging the tilt sensor, since it could be represented with a LED and resistor to demonstrate that the robot was tilted too far. We were also able to gauge whether the PIC reset as the light would flicker every time it rebooted. The element was also successful in making a noise when the battery level reached 0 and the internal structure tilted too far. We deserve 15 points for our ingenuity from using the sound sensor as a debugging tool.

Our manual user input did require extensive outside research. For this, we used the wireless remote controller. There was not sufficient information covered in class or in the textbook for us to do this without outside research. The aspect that we had to look into the most was the signal that was being sent to the receiver. Our research indicated that the signal would be PWM, but our initial tests seemed to be PPM. After more testing and research we found that the signal was in fact PWM, but with duty cycles remaining between about 5 and 9.5 percent. This result complicated how the PIC would receive the signal as the PIC16f88 only has two one pwm port, so we decided to transform the signal inputs to analog with lowpass filters and used two ports' ADC functionality to utilize the signals. For these reasons, we believe that for the manual user input section we deserve the maximum 20 points.

Along with the audio output, our automatic sensor was our other category that required less external research. For the automatic sensor, we used a tilt switch. Although there were some initial issues with the circuitry, we were able to get the tilt switch to successfully send the PIC a high voltage when facing up and a low voltage when at an angle without many complications.

Our fulfillment of the actuators, mechanisms, and hardware category had the most components involved. This was also the section which caused our design to fail. There were the motors with the corresponding circuitry, the ball itself, the internal structure, and the wheels. All of these had to work together, which made this category very complex. Following how the H-bridge should be connected with the PIC and the motors gave us a problem where the PIC would get reset and/or cause inconsistent behavior with the motors running. With outside research, we were able to stop this problem with well placed diodes and capacitors. This did require moderate outside research and the wheels and structures (including the center of gravity) required extensive testing to reach a solution. With the complexity of this category and research and trials that went into it, we believe this section should earn more than 15 points, but possibly not the full 20.

Lastly, the logic, processing, and control also required extensive outside research.

IV. Partial Parts List

The core of this project, Microchip's affordable PIC16F88-I/P, is available for $4.90 from Digikey. This 18-pin microcontroller had just the right amount of digital I/O pins, ADC converters, and CCP modules to suit our project's needs.

The Texas Instruments SN7447AN was used as our BCD to seven-segment display converter. It took a four-bit output from the PIC and converted it to inputs for the LCD display. These through-hole components come in a 16-pin DIP package and can be bought for $3.34 from Digi-Key.

Our DC motors were driven by a L293DNE quadruple half H-bridge chip. Enclosed in a 16-pin DIP package, the L293DNE is available from Mouser for $4.29. It provided a simple yet effective method of controlling bidirectional motion for our two DC motors using logic output.

Our linear voltage regulators were the LM7809C and the LM7805CT. These are rated to 1A and regulate the power coming from our LiPo to the rest of the circuit. The 7809 powered the motors, while everything else was supplied by the 7805. These are available from Mouser for $3.41 apiece.

The transmitter used to control our device was the FlySky FS-I6-M2. This particular transmitter was used for testing because it was already owned, and had a comprehensive suite of settings, tuning options, and control functionality that made it ideal for experimentation and troubleshooting. This controller costs $65 new, but given the functionality present in our final design, a much cheaper and simpler alternative could have been used.

Finally, the receiver used was the FlySky FS2A. It functions at 2.4 GHz and has four channels. It can be configured to output a form of PWM, which was important in our development and testing processes. These receivers are very small, and are available domestically for $16.89 on Amazon, and cheaper from international sources on sites such as eBay.

V. Lessons Learned

As nearly every project group can attest to, our biggest lesson learned involved time management. We started building our circuit, code, and hardware earlier than most groups, but we were still very pressed for time. As soon as ideas are getting finalized, work needs to begin on building the project. Although this can be difficult given concurrent workload in other courses, it would have saved us a lot of stress towards the end of the project.

Additionally, problems should be expected at every step of the process, even with the simplest of subsystems. The best example of this for us was with the seven-segment display. What we assumed was some more complicated issue with the PIC or power management system that frustrated us for hours on end was actually just a mix-up regarding the part number of the display and whether it was common-anode or common-cathode. It is often the smallest, simplest things that cause the biggest headache.

Building on the concept of troubleshooting and debugging, something that we should have incorporated into our project was better user interfacing. While the user only needs to control the device with the transmitter, it's a terrible method for testing outputs as we were never certain if the code that interpreted the transmitter signal was working perfectly. It would have helped massively to integrate a 4-way DIP switch into our design to allow for quick and easy testing of uploaded programs, and potentially additional modal functionality down the road.

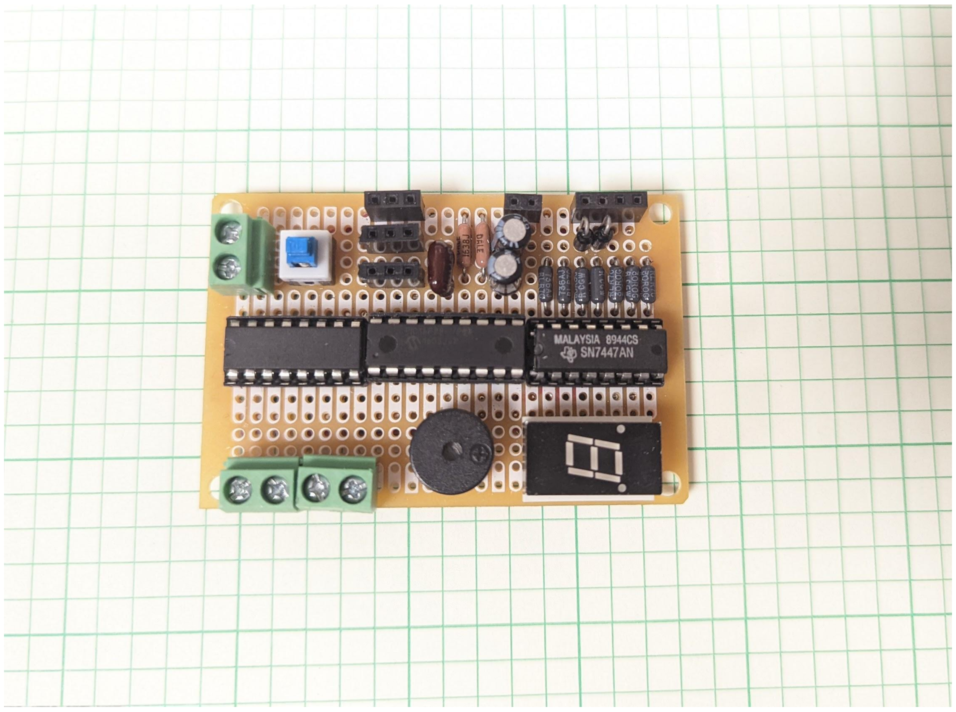
Another lesson we learned was to simplify unnecessarily complicated things. The input signal from our receiver was PWM, and we also needed one of our outputs to be in PWM. This was a huge issue since the PIC we used only had one CCP module, and getting the pulsein function to work had proven very difficult. We spent a significant amount of time trying to find a work-around in the code, to no avail. Eventually, we realized that with a simple RC circuit, we could get an average output voltage from the receiver and send it through an internal ADC. This worked well, and saved us a lot of trouble!

Something that could have saved us hours of work is making new code files whenever trying something new. A specific example of this was when we were working on the PWM output for the motor control. We tried many different techniques, and sometimes used the same file for different approaches. However, we ultimately used parts of an early solution we found, but had overwritten this code. This specific example did not set us back by much, but when errors like this happen often, the time sink becomes significant.

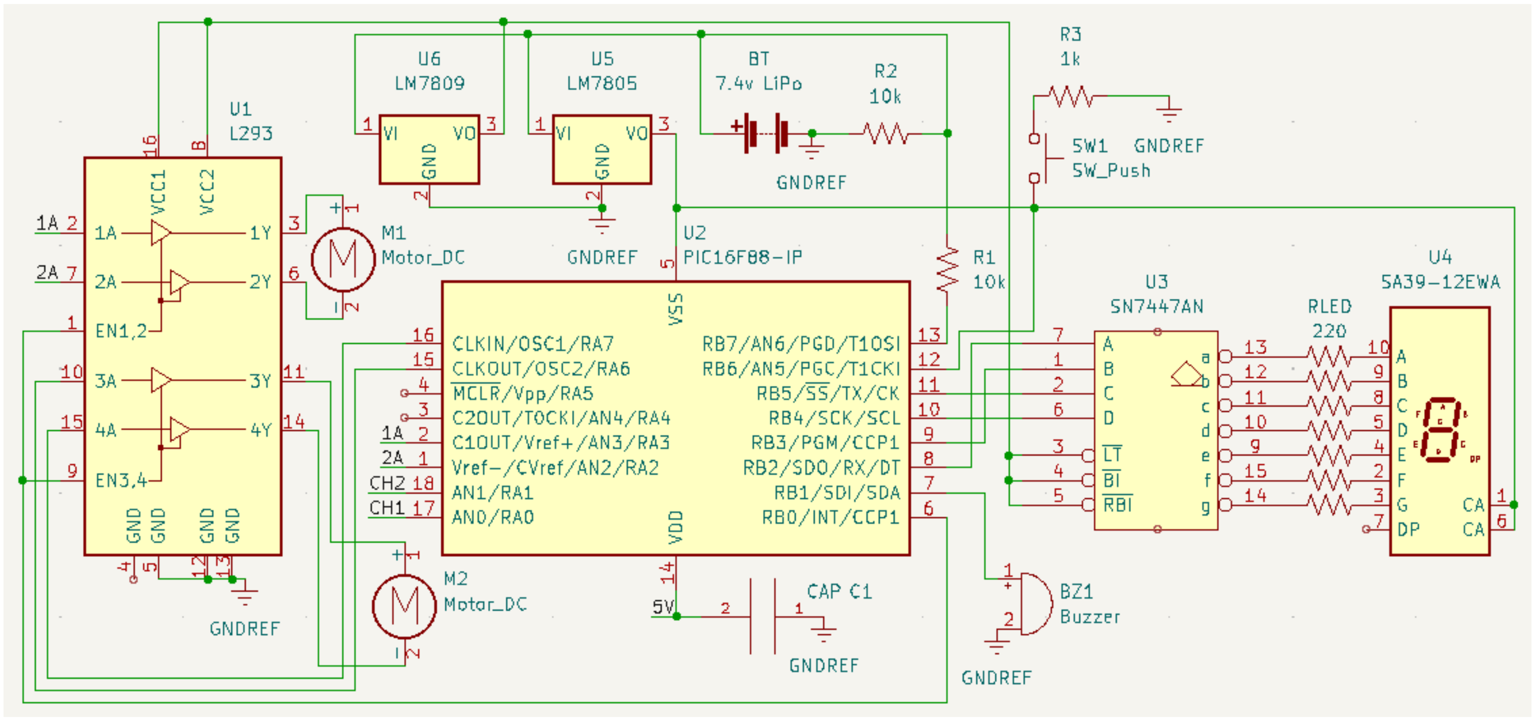
Lastly, it is important not to overreach with the design. We chose a project that we knew would be ambitious. From our initial design, we did lessen the complexity by eliminating the camera module and its stabilization system. However, the overall design was still too ambitious given the amount of time we had to complete it. With this being said, choosing a project at this level of complexity allowed us to learn more than we would have otherwise, and made it all the more satisfying to see the final product in action.

VI. Appendix

Photograph of Final Soldered Platform Board, Partially Depopulated



Complete KiCad Project Circuit Schematic



Final Demonstration Code

```
'Name    : finalcode.BAS
'Authors : Clyde J.
'        : Matthew R.
'        : Max U.
'Date    : 5/16/2022
'Version : 2.0
'Notes   : Remote-Controllable Robotics Platform
'        : Final Project Demo Code

' Configuration for PIC setup without external oscillator
' or MCLR pin reset function enabled
#config
    __CONFIG _CONFIG1, _INTRC_IO & _PWRTE_ON & _MCLRE_OFF & _LVP_OFF
#endconfig

' Set up ADC functionality for proper size and clock speed
DEFINE  ADC_BITS        10      ' Set number of bits in result
DEFINE  ADC_CLOCK       3       ' Set clock source (3=rc)
DEFINE  ADC_SAMPLEUS    15      ' Set sampling time in uS

' Set up internal oscillator at 8MHz
define osc 8
OSCCON.4 = 1 : OSCCON.5 = 1 : OSCCON.6 = 1

' Define variables for 7-segment binary input
sevend var PORTB.4
sevenc var PORTB.5
sevenb var PORTB.2
sevena var PORTB.3

speaker var PORTB.1 ' Speaker output variable
tilt var PORTB.6    ' Tilt switch input variable

' PWM and motor direction output variables
pwmsig var PORTB.0
fowardleft var PORTA.2
backwardleft Var PORTA.3
fowardright var PORTA.7
backwardright var PORTA.6

‘ Configure port tristate buffers for inputs/outputs
TRISA = %00000011
TRISB = %11000000

‘ Configure ANSEL register to use ADC at AN0, AN1, AN7
ANSEL = %1000011

‘ Configure CCP module and time period register for PWM out
CCP1CON = %00001100
T2CON = %00000110
PR2 = 124

' Program variables
battlevel var word     ‘ Adjusted Battery level
adcbatt var word       ‘ Battery level ADC input
duty VAR WORD          ‘ PWM out duty cycle
FBsig var word         ‘ CH1 in variable
RLsig var word         ‘ CH2 in variable
```

```
‘ Motor control variables
FBwidth var word
RLwidth var word
ch1 var word
ch2 var word
ch1var var word
ch2var var word
ch1varH var word
ch2varH var word
ch1varL var word
ch2varL var word

‘ Receiver input averaging loop creates “neutral setpoint”
‘ on platform startup that can be referenced later
i var byte

ch1var = 0
ch2var = 0
pause 1000
for i = 0 to 4
    adcin 1, ch1
    adcin 0, ch2
    ch1var = ch1var + ch1
    ch2var = ch2var + ch2
    pause 300
next
ch1var = ch1var / 5
ch2var = ch2var / 5

‘ Set desired duty cycle and calibration widths for the
‘ “bounds” of receiver input interpretation
duty = 800
FBwidth = 33
RLwidth = 33
ch1varL = ch1var - FBwidth
ch2varL = ch2var - RLwidth
ch1varH = ch1var + FBwidth
ch2varH = ch2var + RLwidth

‘ Set PWM value and ensure speaker is LOW
gosub pwmset
low speaker

‘ Program loop
While(1)

    ' Take input from CH1 and CH2
    ' Forward/Back
    adcin 1, FBsig
    ' Left/Right
    adcin 0, RLsig

    gosub ALARM                       ‘ Check tilt alarm condition
    gosub BATT                        ‘ Update battery indicator

    ‘ Check for forward condition, then turning conditions
    if FBsig > ch1varH then
        if RLsig < ch2varL then
            gosub LEFT
        elseif RLsig > ch2varH then
            gosub RIGHT
        else
            gosub FWD
        endif
    endif

    ‘ Check for backwards condition, then turning conditions
    if FBsig < ch1varL then
        if RLsig < ch2varL then
            gosub LEFT
        elseif RLsig > ch2varH then
            gosub RIGHT
        else
            gosub BACK
        endif
    else
        gosub HALT                    ‘ Halt if no conditions met
    endif
wend

' Sets up CCP1 module to output PWM signal to H-bridge
PWMSET:
    CCP1CON.4 = Duty.0   ' Store duty to registers as
    CCP1CON.5 = duty.1   ' a 10-bit word
    CCPR1L = duty >> 2
return

' Unused - Allows for slow increase in PWM to desired duty
' cycle to mitigate EMF spike during motor start
PWMRAMP:
      duty = 0
      for i = 0 to 4
        duty = duty + 100
        pause 100
      next
return

' Five motor state functions control the H-bridge
FWD:
    high fowardleft
    high fowardright
    low backwardleft
    low backwardright
    pause 100
return

BACK:
    low fowardleft
    low fowardright
    high backwardleft
    high backwardright
    pause 100
return

LEFT:
    high fowardleft
    low fowardright
    low backwardleft
    high backwardright
    pause 100
return
```

```
RIGHT:
    low fowardleft
    high fowardright
    high backwardleft
    low backwardright
    pause 100
return

HALT:
    low fowardleft
    low fowardright
    low backwardleft
    low backwardright
return

' Checks tilt sensor condition when called, activates buzzer
ALARM:
if tilt = 1 then
        high speaker
        pause 50
        low speaker
    elseif tilt = 0 then
        low speaker
    endif
return

' Battery monitor function takes in ADC value of
' Li-Po and calls different 7-segment display states
BATT:
adcin 6, battlevel
low sevend
if battlevel > 1020 then
    gosub BATT5
elseif battlevel < 1020 then
    gosub BATT4
elseif battlevel < 1000 then
    gosub BATT3
elseif battlevel < 900 then
    gosub BATT2
elseif battlevel < 800 then
    gosub BATT1
else
    gosub batt0
endif
return

' Functions for each possible display state
BATT5:
    high sevena
    low sevenb
    high sevenc
return

BATT4:
    low sevena
    low sevenb
    high sevenc
return

BATT3:
    high sevena
    high sevenb
    low sevenc
return

BATT2:
    low sevena
    high sevenb
    low sevenc
return

BATT1:
    high sevena
    low sevenb
    low sevenc
return

BATT0:
    low sevena
    low sevenb
    low sevenc
return
```