Prelab Report: Operational Amplifier Design

Arye Mindell
Loyola Marymount University, Department of Electrical Engineering and Computer Science

***Abstract*—This prelab report presents the design and simulation of an operational amplifier (op-amp) that meets specified gain, input resistance, and output impedance constraints. Simulations are conducted to verify key parameters, including gain, bandwidth, and voltage swing.**

I. Introduction

This lab involves designing a two-stage op-amp with a differential amplifier input stage, ensuring high input impedance and controlled gain, followed by a gain stage that enhances the voltage amplification. The design must meet the following constraints:[1]

- Small-signal voltage gain: $400 \leq A_d \leq 500$
- Input resistance: $R_{id} \geq 20k\Omega$
- Output resistance: $R_o \leq 200\Omega$
- Load resistance: $R_L = 200\Omega$
- Low-frequency cutoff: $f_{L3dB} \leq 100Hz$
- Maximum peak-to-peak output swing: $V_{o,p-p} \geq 2V$
- Offset voltage: $|V_{\text{offset}}| \leq 0.01V$

II. System Description

The operational amplifier consists of multiple stages designed to achieve high gain, low output impedance, and stability. The key components include a differential amplifier, a common-emitter gain stage, a level shifter, and an emitter follower output stage. The complete circuit layout is shown in Fig. 1.

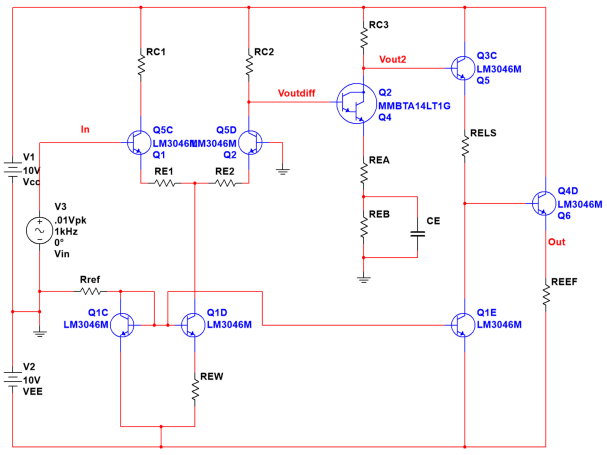


Fig. 1. Circuit layout of the designed operational amplifier.

1) **Differential Amplifier:** The first stage is a differential amplifier, which serves as a primary gain stage while also filtering common-mode noise. This stage is designed to provide high input impedance and a stable reference for signal amplification. A Widlar current source is used to bias this stage, ensuring a predictable and stable operating point.
2) **Common-Emitter Stage:** The second stage consists of a common-emitter (CE) amplifier, which provides additional voltage gain. A Darlington pair configuration is used to isolate the output resistance of the CE stage from the next stage, preventing excessive loading effects.
3) **Level Shifter:** A level shifter stage is introduced to remove the DC offset caused by the biasing of the previous stages. This ensures that the output voltage remains centered around the desired reference level, preventing distortion and maximizing the usable output swing. This stage is biased by a standard current mirror, sharing a reference current with the Widlar biasing of the first stage.
4) **Emitter Follower Output Stage:** The final stage is an emitter follower, which significantly reduces the output resistance, ensuring that the amplifier can drive low-impedance loads effectively while maintaining the designed gain characteristics.

III. Design

A. Biasing Currents

To ensure that the output swing at the emitter follower exceeds $1Vpk$, the small-signal voltage at the emitter of Q_5 must be greater than 1V, as given by:

$$I_{E5}(R_{E5} \parallel R_L) \geq 1V \tag{1}$$

The value of R_{E5} must be significantly larger than 200Ω in order to achieve a DC voltage drop of 10V from the emitter to VEE with a practical supply current. Assuming $R_{E5} \geq 200\Omega$:

$$\frac{1V}{200\Omega \parallel 200\Omega} = \frac{1V}{100\Omega} = 10mA \leq I_{E5} \tag{2}$$

To provide a safety margin for output swing,

$$I_{E5} = 12.5mA \tag{3}$$

Due to the high input resistance requirement, the differential pair emitter current I_{E1} will be in the microamp range, giving a total I_{EE} of $\approx 1mA$.

The level shifter must produce a significant voltage drop across its emitter resistor to compensate for the DC shift introduced by prior stages. The resistor R_{E4} also affects the output resistance by being reflected through Q_5 and scaled by a factor of β. Thus, I_{E4} must be large enough to generate a substantial voltage drop across R_{E4} while keeping its resistance in the 10kΩ range. Since the level shifter is biased

by a current mirror, I_{E4} must match the reference current I_{REF}, and both must share the remaining current budget of 6.5mA with the biasing current of the CE stage.

$$I_{E4} = I_{REF} = 2mA \tag{4}$$

I_{E4} and I_{REF} are set to 2mA, leaving 2.5mA for biasing the CE stage.

The current I_{C3} in the CE stage influences both gain and maximum symmetrical swing. The gain can be adjusted by modifying R_{EA}; therefore, I_{C3} is chosen to ensure adequate voltage swing. To enable a swing of at least 1V at the output:

$$\frac{V_{swing,out}}{A_{vLS} \times A_{vEF}} = \frac{1}{0.95 \times .75} = 1.4V \leq V_{swing,CE} \tag{5}$$

the CE stage must provide a swing of at least 1.5V to account for the gain of the final two stages while maintaining headroom before distortion. However, the voltage drop across R_{C3} must remain moderate to preserve the differential stage's available swing, since the two stages are DC-coupled. Additionally, I_{C3} must be sufficiently large to prevent an excessively high R_{C3}, which would require an impractically large R_{EA} to stay within the gain constraints.

Setting $V_{R_{C3}} = 2V$ and selecting $I_{C3} = 1mA$ results in:

$$R_{C3} = \frac{V_{R_{C3}}}{I_{C3}} = \frac{2V}{1mA} = 2k\Omega \tag{6}$$

This sets the voltage swing and current, while the gain can be adjusted later using the emitter resistance.

B. Differential Stage

Neglecting the output resistance of the current source, the input resistance of the differential stage is approximated as:

$$R_{id} = 2(\beta + 1)(r_e) \tag{7}$$

Assuming $\beta = 100$ and setting the input resistance requirement to $R_{id} \geq 20k\Omega$, the minimum total emitter resistance r_e is given by:

$$r_e = \frac{R_{id,min}}{2(\beta + 1)} = \frac{20k\Omega}{2(100 + 1)} = 99\Omega \tag{8}$$

Solving for the emitter current using $I_E = V_T/r_e$, where $V_T \approx 26mV$ at room temperature:

$$I_E = \frac{26mV}{99\Omega} = 262\mu A \tag{9}$$

A lower biasing current limits the maximum output voltage swing, so I_E is chosen to preserve the output swing as much as possible. R_E is chosen to meet the input resistance requirement with a safety margin.The final values selected are:

$$I_{E1,2} = 200\mu A, \quad r_{e1,2} = 130\Omega \tag{10}$$

$R_{C1,2}$ is selected to set the biasing voltage at the output of the differential amplifier, $V_{C1,2} = V_{C3} - 1.5$, Assuming a 1.5V swing is needed for the common emitter stage.

$$R_{C1,2} = \frac{V_{CC} - V_{C1,2}}{I_{C1,2}} = \frac{10V - 6.5V}{200\mu A} = 17.5k\Omega \tag{11}$$

To allow for precise tuning of the voltage gain without radically changing the design, emitter-degenerating resistors are used. Their values are determined to set the differential gain to approximately 30, which provides a good base for the total gain while limiting the necessary stage output swing to $1.5V/30 = 50mV$.

The differential gain is given by:

$$A_{vd} = \frac{R_{C1,2}}{2(r_e + R_{E1,2})} = 30V/V \tag{12}$$

Solving for $R_{E1,2}$:

$$R_{E1,2} = \frac{R_{C1,2}}{2A_v} - r_e = 160\Omega \tag{13}$$

C. Current Source

The Widlar current source is designed to set the tail current as:

$$I_{EE} = 2I_{E1,2} = 400\mu A \tag{14}$$

since I_{REF} is matched to I_{E4}:

$$I_{REF} = 2mA \tag{15}$$

These values determine the resistances of R_E and R_1:

$$R_E = \frac{V_T \ln(I_{REF}/I_o)}{I_o} = 104\Omega \tag{16}$$

$$R_1 = \frac{0V - V_{BE1} - V_-}{I_{REF}} = 4.7k\Omega \tag{17}$$

D. Common-Emitter Stage

Assuming the total gain required is 450 V/V, the gain required from the common-emitter stage is:

$$A_{v2} = \frac{450}{(30 \times 0.7 \times 0.95)} = 22V/V \tag{18}$$

Using the designed values for R_{C3} and I_{C3}, R_{EA} is chosen to achieve the required gain:

$$A_{v2} = \frac{R_{C3}}{\frac{2V_T}{I_{C3}} + R_{EA}} \tag{19}$$

Substituting $R_{C3} = 2k\Omega$, $A_{v2} = 22$, $V_T \approx 26mV$, and $I_{C3} = 1mA$:

$$R_{EA} = \frac{2000\Omega}{22} - \frac{2(26mV)}{1mA} \tag{20}$$

$$R_{EA} = 90.9\Omega - 52\Omega = 38.9\Omega \tag{21}$$

The total emitter resistance is given by applying KVL from the collector of Q_2 through the darlington transistor and to ground, assuming $V_{BE,\text{darl}} = 1.3V$:

$$R_{E,\text{total}} = \frac{V_{C1} - V_{BE,\text{darl}}}{I_{C3}} \tag{22}$$

$$= \frac{6.5V - 1.3V}{1mA} = \frac{5.2V}{1mA} = 5.2k\Omega \tag{23}$$

R_{EB} is the difference between R_{EA} and $R_{E,\text{total}}$:

$$R_{EB} = R_{E,\text{total}} - R_{EA} = 5.2k\Omega - 38.9\Omega = 5.16k\Omega \quad (24)$$

The resistance seen by the capacitor is:

$$R_{CE} = R_{EB} \parallel \left(R_{EA} + \left(\frac{V_T}{I_{C4}} + \frac{R_{C1,2}}{\beta}\right)\right) \quad (25)$$

$$R_{CE} = 5.2k \parallel \left(40 + \left(\frac{0.026}{1mA} + \frac{17.5k}{10000}\right)\right) \quad (26)$$

$$R_{CE} = 5200 \parallel 62.5\Omega = 66.6\Omega \quad (27)$$

Estimating the resistance seen by the capacitor as equal to REB, the 3dB cutoff frequency is:

$$f_{L,3dB} = \frac{1}{2\pi C_E R_{CE}} \quad (28)$$

Substituting $f_{L,3dB} = 100$ Hz and solving for C_E:

$$C_E = \frac{1}{2\pi \times 100 \times 66.66} = 23.6\mu F \quad (29)$$

Therefore a standard $22\mu F$ capacitor is used.

E. Level Shifter and Emitter-Follower

The emitter resistance of the level shifter stage, R_{E4}, is chosen to set the DC offset at the output to 0. The voltage across R_{E4} and the emitter current I_{E4} are known, allowing the resistance to be determined as:

$$R_{E4} = \frac{V_{C3} - V_{BE4} - V_{BE5}}{I_{E4}} \quad (30)$$

Substituting the values $V_{C3} = 8V$, $V_{BE4} = V_{BE5} = 0.7V$, and $I_{E4} = 2mA$:

$$R_{E4} = \frac{8V - 0.7V - 0.7V}{2mA} = \frac{6.6V}{2mA} = 3.3k\Omega \quad (31)$$

Similarly, since $V_{R_{E5}}$ and I_{E5} are known, the resistance R_{E5} is determined using Ohm's law:

$$R_{E5} = \frac{V_{R_{E5}}}{I_{E5}} = \frac{10V}{12.5mA} = 800\Omega \quad (32)$$

IV. Design Check

To validate the design, the expected gain and input resistance values are calculated.

The exact equation for the input resistance is:

$$R_{id} = 2 * (\beta + 1)\,(R_{E1,2} + r_e)) \quad (33)$$

Substituting $r_e = 130\Omega$ and $\beta = 100$:

$$R_{id} = 101 \times (130\Omega + 160\Omega) = 58.6k\Omega \quad (34)$$

Since $R_{id} \geq 20k\Omega$, this satisfies the requirement.

The total gain is the product of the individual stage gains:

$$A_v = A_{v,d} \times A_{v,CE} \times A_{v,LS} \times A_{v,EF} \quad (35)$$

Neglecting the Early effect, the gain of the level shifter stage is given by:

$$R_{B5} = (\beta + 1) \times ((R_{E5} \parallel R_L) + r_{e5}) \quad (36)$$

Substituting:

$$R_{B5} = (101) \times ((800\Omega \parallel 200\Omega) + 2\Omega) = 16.362k\Omega \quad (37)$$

$$A_{v,LS} = \frac{R_{E4}}{r_{e4} + R_{E4}} \times \frac{R_{B5}}{R_{E4} + R_{B5}} \quad (38)$$

Substituting:

$$A_{v,LS} = 0.996 \times 0.832 = 0.828V/V \quad (39)$$

The gain of the Emitter-Follower stage is given by:

$$\frac{R_{E5}}{r_{e5} + R_{E5}} = 0.997V/V \quad (40)$$

Substituting the stage gains into (30):

$$A_v = 30 \times 22 \times 0.828 \times 0.997 = 541V/V \quad (41)$$

This does not satisfy the requirement, but the emitter degenerating resistors allow for the gain to be adjusted experimentally. Therefore no compensation is necessary in the design. Potentiometers will be used in the lab to implement precise gain adjustment. The output resistance is:

$$R_{out} = R_L \parallel R_{E5} \parallel \left(\frac{V_T}{I_{C5}} + \frac{R_{E4}}{\beta}\right) \quad (42)$$

$$= 200 \parallel 800 \parallel \left(\frac{0.026}{0.0125} - \frac{3.3k\Omega}{100}\right) = 29\Omega \quad (43)$$

Since $R_{out} \leq 200\Omega$, this satisfies the requirement.

The peak output voltage is:

$$V_{swing} = (V_{CG} - V_{C4}) \times A_{LS} \times A_{EF} \quad (44)$$

$$= (10 - 8) \times 0.828 \times 0.997 = 1.65V \quad (45)$$

Since $V_{swing} \geq 1V$, this satisfies the requirement. The maximum input peak-to-peak voltage is:

$$V_{ip-p} = \frac{V_{o,p-p}}{A_{\text{total}}} = \frac{2}{541} = 6mV \quad (46)$$

The resistance seen by the capacitor is:

$$R_{CE} = R_{EB} \parallel \left(R_{EA} + \left(\frac{V_T}{I_{C4}} + \frac{R_{C1,2}}{\beta}\right)\right) \quad (47)$$

$$R_{CE} = 5.2k \parallel \left(40 + \left(\frac{0.026}{1mA} + \frac{17.5k}{10000}\right)\right) \quad (48)$$

$$R_{CE} = 5200 \parallel 62.5\Omega = 66.6\Omega \quad (49)$$

The 3dB cutoff frequency is:

$$f_{L,3dB} = \frac{1}{2\pi C_E R_{CE}} = \frac{1}{2\pi \times 22\mu F \times 66.6} = 108.6Hz \quad (50)$$

this does not meet the specification, so a larger capacitor may be needed.

V. Simulation Results

To validate the design, various simulations were performed, including DC operating point analysis, AC gain analysis, input impedance measurement, and transient response analysis. The results are presented below.

A. DC Operating Conditions

The DC biasing conditions of the circuit were verified to ensure proper transistor operation. The measured DC voltages and currents confirm that the circuit operates within the expected parameters, and show that the DC offset voltage at the output is less than $10mV$

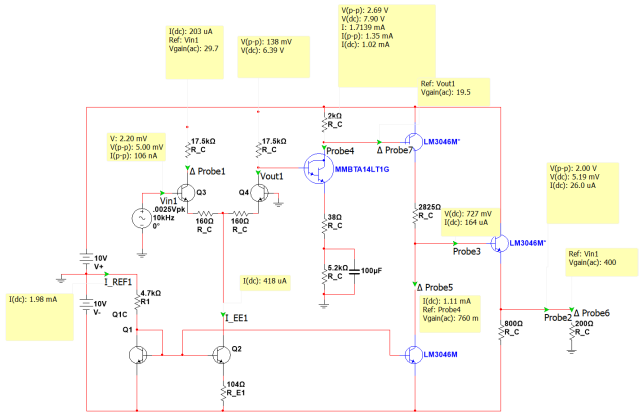


Fig. 2. Simulated DC operating conditions of the amplifier.

B. AC Gain and Frequency Response

The open-loop gain and bandwidth of the amplifier were analyzed using an AC sweep. The frequency response plot in Fig. 3 shows the midband gain of 467.87 V/V which satisfies specifications. The low-frequency cutoff occurs at about $45Hz$, which satisfies the specification.

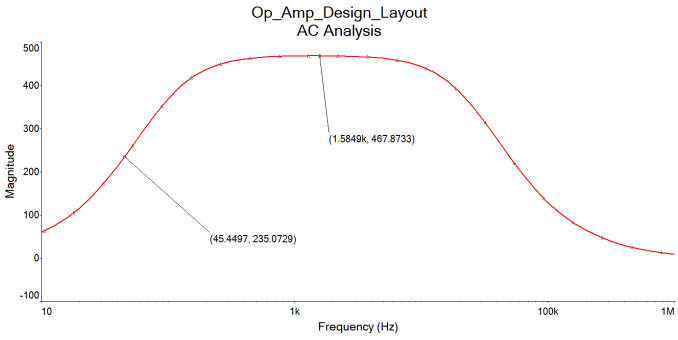


Fig. 3. Simulated AC gain and frequency response.

C. Input Impedance Measurement

The input impedance of the amplifier was measured at 1 kHz using an AC analysis. The results, shown in Fig. 4, indicate an input resistance of 47.15 kΩ, which satisfies specification.

Op_Amp_Design_Layout
Single Frequency AC Analysis @ 1000 Hz

	Variable	Real	Imaginary
1	V(Vin1)/I(Vin1)	47.15007 k	-224.94779

Fig. 4. Simulated input impedance at 1 kHz.

D. Transient Response Analysis

A transient analysis was performed to examine the time-domain behavior of the amplifier. The output waveform in Fig. 5 shows a peak-to-peak voltage swing of approximately 2.1V without distortion, which meets specification.

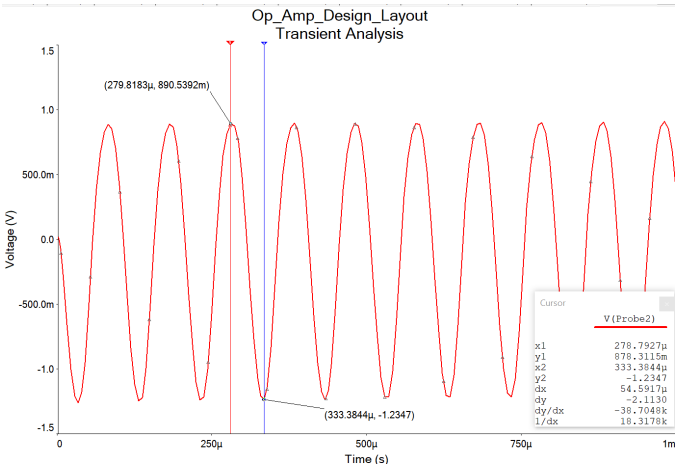


Fig. 5. Simulated transient response of the amplifier.

These simulation results validate the design, ensuring it meets the required specifications for gain, bandwidth, input impedance, and output voltage swing.

A comparison between the designed and simulated values is provided in Table I. This table summarizes key parameters, including gain, impedance, cutoff frequency, and bias currents, to verify that the circuit meets the design specifications. The simulated values closely match the designed values, confirming that the amplifier performs as expected.

TABLE I
Comparison of Designed and Simulated Values

Parameter	Designed Value	Simulated Value
$A_{v,d}$	30	29.7
$A_{v,CE}$	22	19.5
$A_{v,LS}$	0.7	0.828
$A_{v,EF}$	0.99	0.99
$A_{v,total}$	450	467.87
R_{in}	$\geq 20k\Omega$	47.15kΩ
R_o	$\leq 200\Omega$	29Ω
$f_{L,3dB}$	≤ 100 Hz	45.6 Hz
$V_{o,peak}$	$\geq 1V$	1.39V
V_{ip-p}	≥ 6 mV	6.5 mV
I_{EE}	400 μA	418 μA
$I_{C1,2}$	200 μA	203 μA
I_{C3}	1 mA	1.02 mA
I_{E4}	2 mA	1.99 mA
I_{E5}	12.5 mA	12.7 mA
V_{C1}	6.5 V	6.39 V
V_{C3}	8 V	7.90 V
V_{E4}	700 mV	727 mV
V_{E5}	$\leq 10mV$	5.19 mV

VI. Expected Results

The implemented circuit is expected to demonstrate:

- An open-loop gain between 400 and 500.
- An input resistance greater than 20kΩ.
- An output resistance below 200Ω.
- A low-frequency cutoff within specifications.
- A peak-to-peak output swing of at least 2V.

VII. Conclusion

This prelab outlines the theoretical design and expected performance of a two-stage operational amplifier. Simulations will validate the design against the required specifications, ensuring its practical feasibility.

References

[1] Loyola Marymount University, Department of Electrical Engineering and Computer Science, "EECE3200 Lab 6: Operational Amplifier Design," 2025.

URBAN TURTLE

ENES100: OSU PROTOTYPE REVIEW

PROJECT DETAILS: WATER

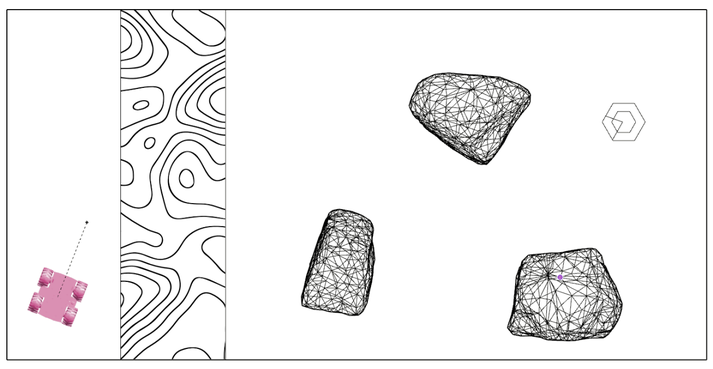
OBJECTIVES

> Navigate to mission site

> Transmit water type

> Collect small water sample

> Transmit pool depth

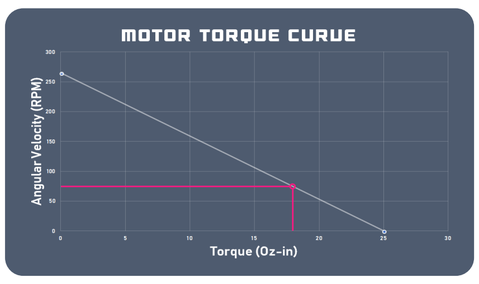


GEN. CONSTRAINTS

> Max weight 3 kg

> 350mm x 350mm footprint

> Replication cost < $350



INITIAL SCHEDULE

> March 6—Prototype Complete

> March 17—Sensor Rig Complete

> April 21—Final CAD Shipped

> May 7—Final Demonstration Passed

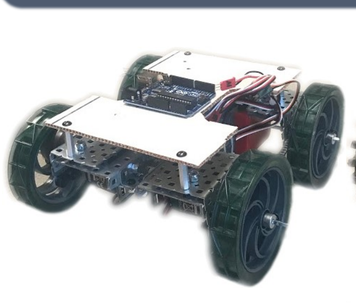
OSU DETAILS

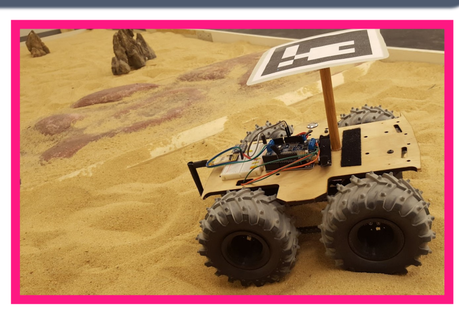
> Motors spec'd to use ~70% of torque when turning @ 170 RPM

> Battery (2000 mAh) lasts for 20 mins at full current draw (~6A)

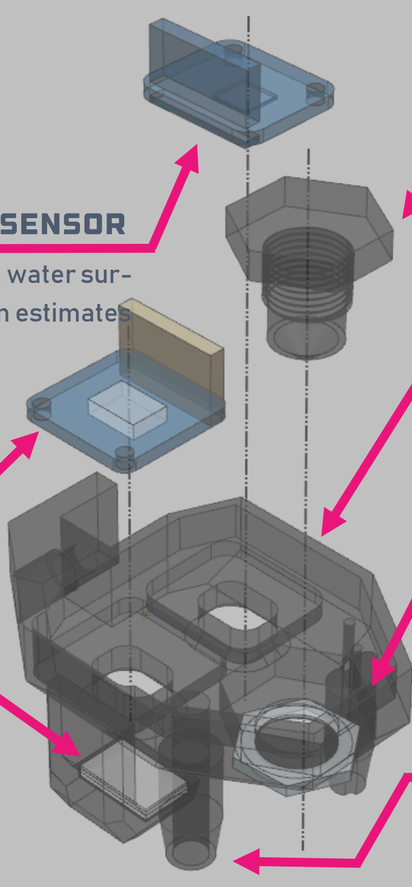
> Final weight: 2.3kg

> Final BOM Cost: $300





SENSOR BREAKDOWN



LASER DISTANCE SENSOR

Measures distance from water surface to pool bottom, then estimates depth

COLOR SENSOR

Detects the water color to determine if polluted

COLOR MIRROR

Reflects water color when submerged to help sensor detect pollution

WATER SENSOR

Detects water surface and stops rig lowering

SENSOR RIG

3D Printed housing for all sensors

CONDUCTIVITY PROBE

Determines if water is salty by EC value

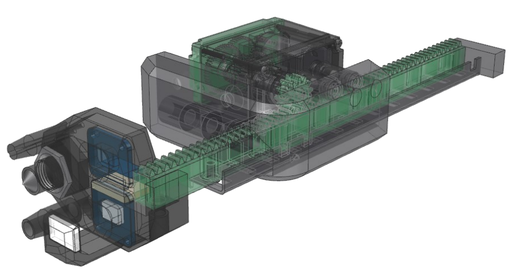
PUMP GUIDE

Carries the pump stem below the surface to gather a water sample

DESIGN

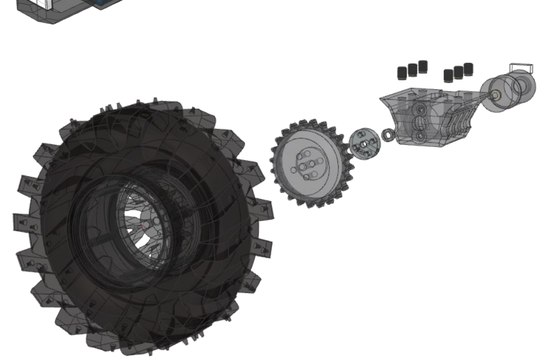
SENSOR RIG

- Carries sensors and pump stem below water surface
- Designed to operate on a hinge for packaging
- Motor, rack and pinion actuation system



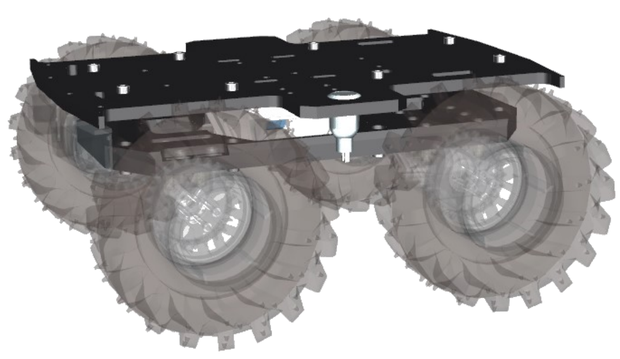
DRIVETRAIN

- Differential Chain Drive
- Custom 3D-printed axle assembly
- Estimated speed over sand: 0.65 m/s



STRUCTURE

- CNC'ed UHMW Baseplate
- Lasercut wood electronics board
- Access hole for battery replacement
- Threaded inserts for secure mounting



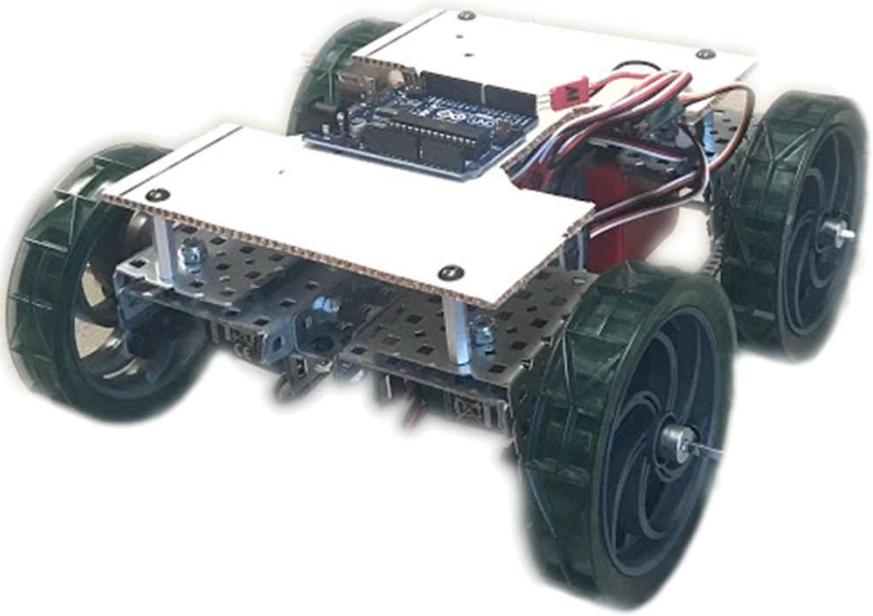
OSU PERFORMANCE

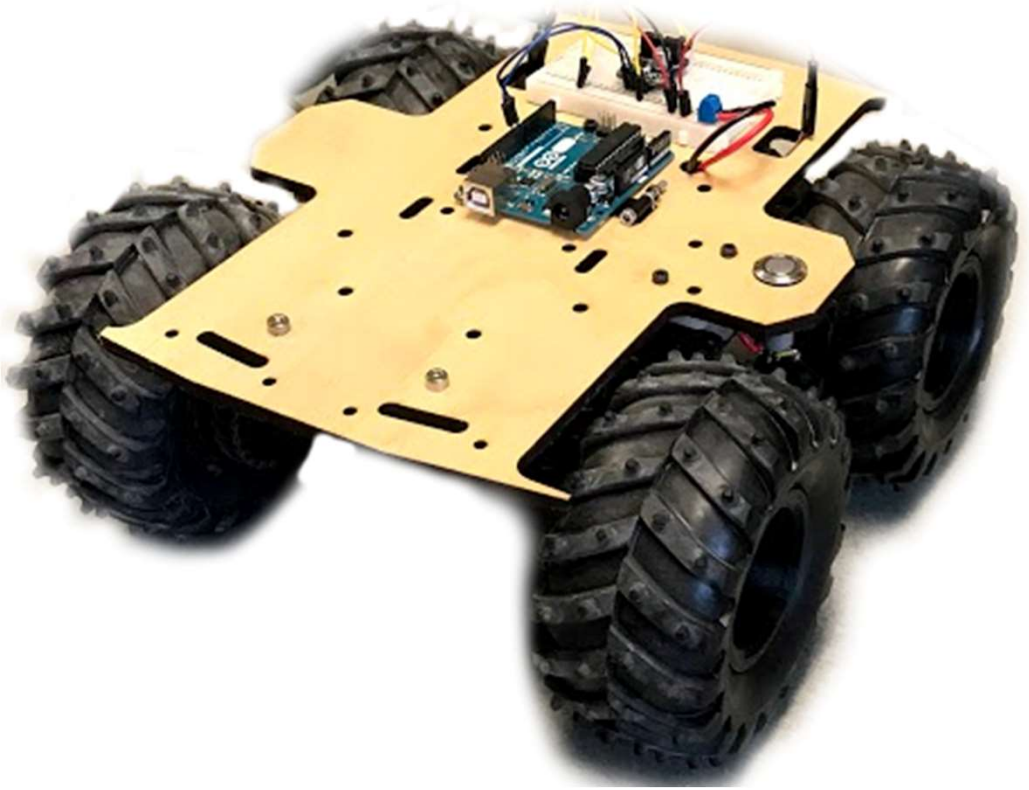
- Successfully navigated over rocky terrain
- Successfully lowered sensor rig and detected water type
- Failed to accurately navigate to the mission site
- Failed to collect water sample (pump was never installed)

LESSONS LEARNED

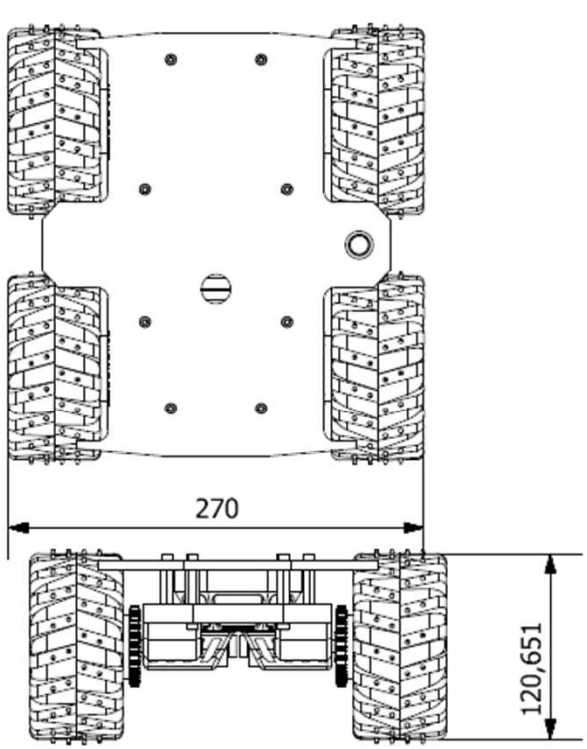
- Turning torque is a much tighter constraint than driving torque
- Electronics requires the most troubleshooting time in robot prototyping
- Communicating work done is crucial to production efficiency
- When creating initial design schedules, you have to consider shipping lead time

Vehicle Prototypes

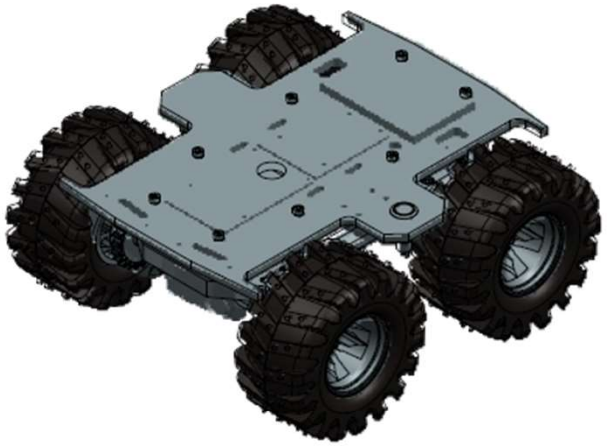


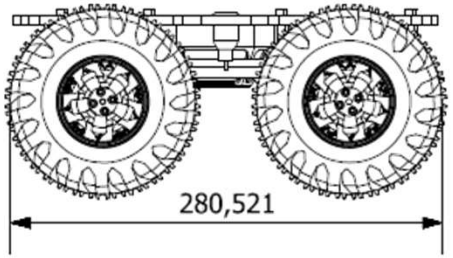


Preliminary Design Details

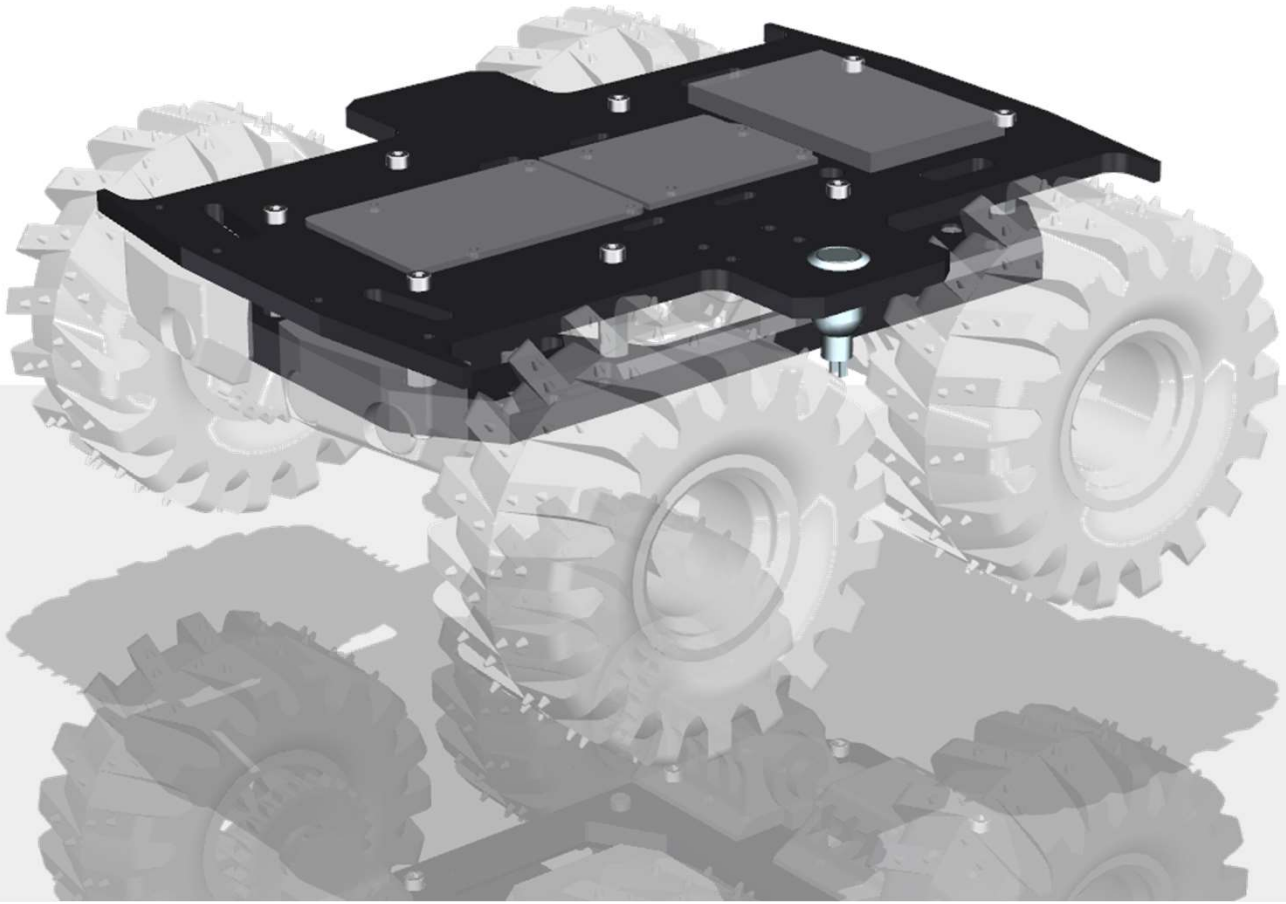


SCALE 1:4



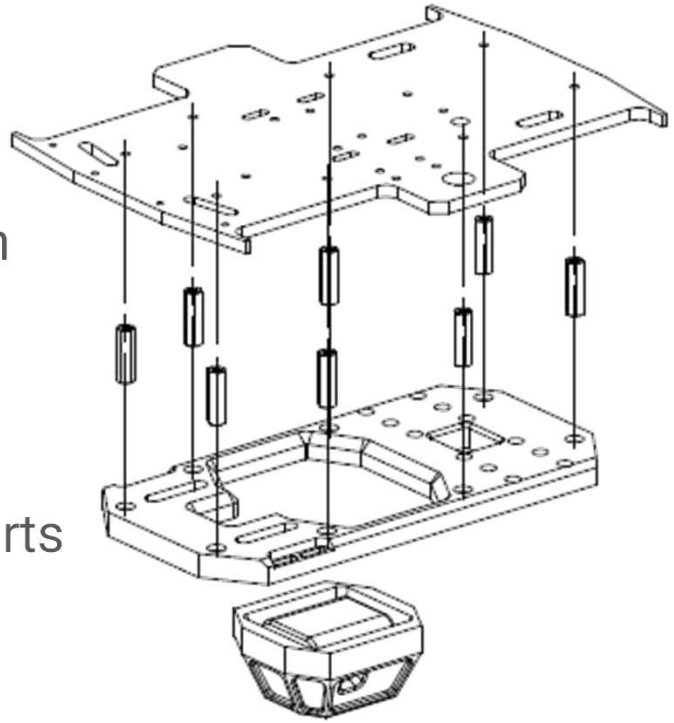


Structure



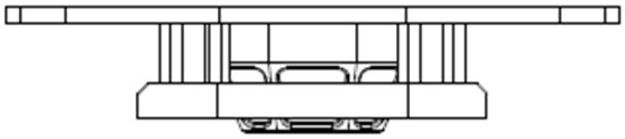
Structure

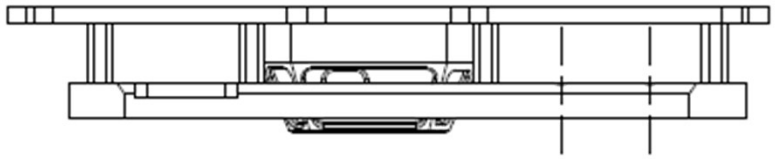
- Thick UHMW base-plate for chassis strength
- Easy removal of electronics
- Mounting slots for chain tensioning
- Access hole for battery replacement
- Use threaded inserts to mount 3D printed parts



Updates

- Add protective cover for electronics
- Battery holder mounting interface
- Find cheaper alternative to standoffs

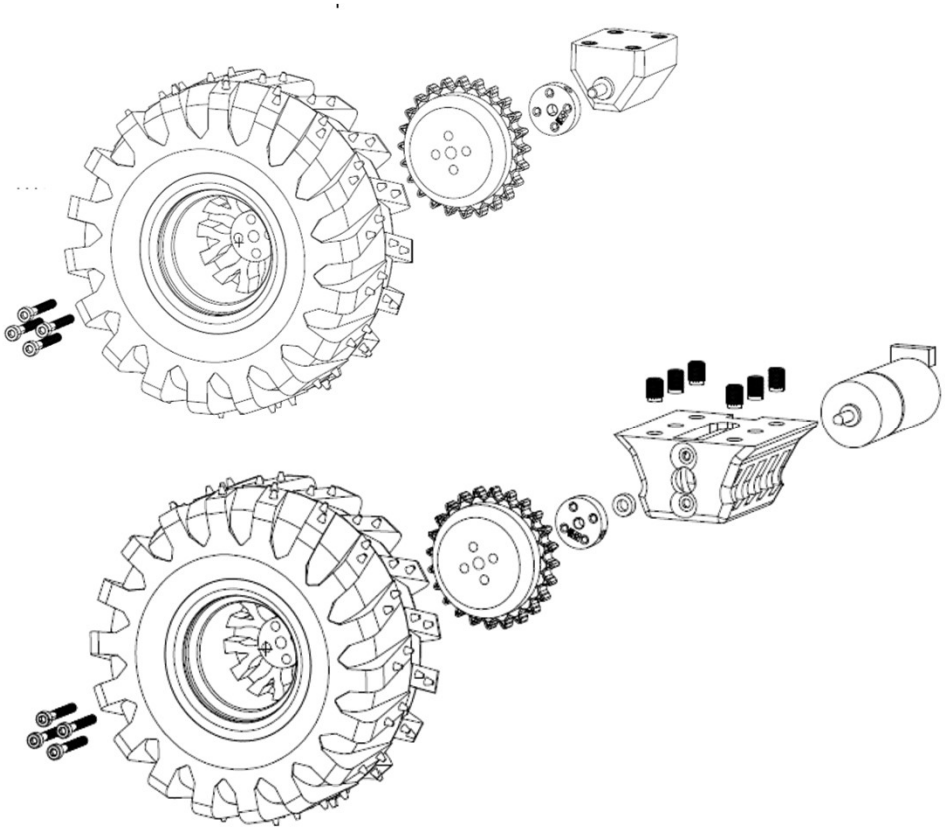




Propulsion

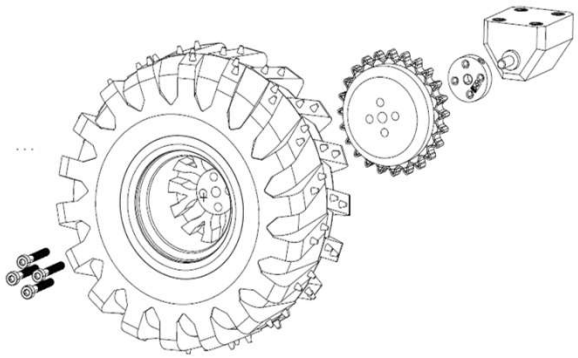
Propulsion

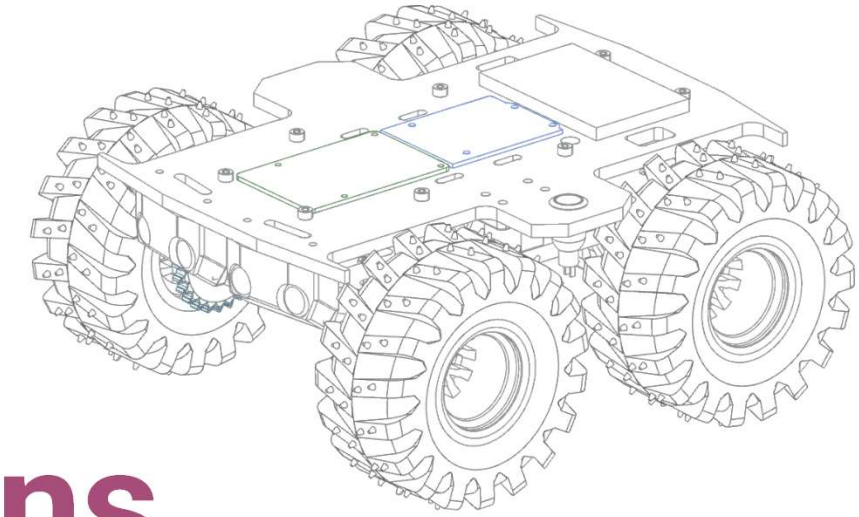
- Chain drive
- Max speed 1.5 m/s
- 120mm Dia. tires
- Predicted current draw - 0.79A / motor
- 36% of stall torque



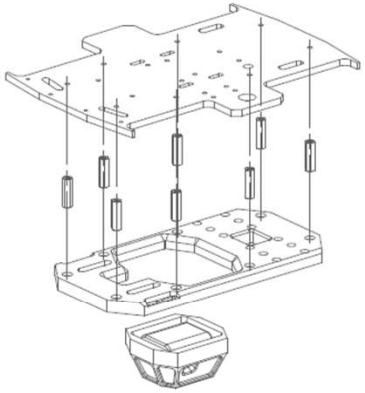
Updates

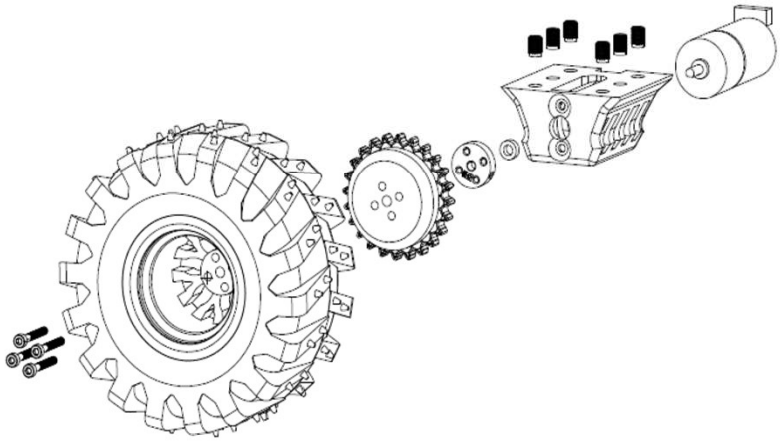
- Source a higher torque motor
  - Failed to account for turning scrub
  - Rolling resistance calc was way off
- Tighten hole tolerances (slop)
- Chain works awesome!

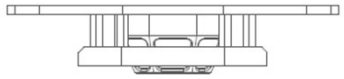


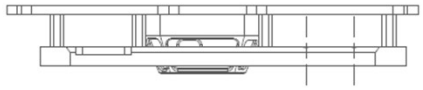


Questions









TEAM 1389 STRATUS Software Overview

Arye Mindell, Rafael Metz

Features

- **Reuseable code repository** - many complex tasks on the robot are handled by *Ohm*, an extensive library of reusable code we wrote as a base for all future robot projects
- **Computer vision** - the robot uses an image processing routine to identify the airship's lift peg, align with it, and place gears on it accurately
- **Autonomous path following** - during the autonomous period, the robot can drive in curved paths to reach its destination in the most efficient way possible
- **Web dashboard** - during matches, a sleek dashboard interface offers a comprehensive readout of the robot's systems, and notifies the drivers of any robot failures as they come up
- **Simulation** - a custom physics simulator enables us to test any code written using our library, Ohm, without a working robot available

Team 1389's reusable robot code library: Ohm

Ohm is a java library for FRC robots. It is intended to augment the features of WPILib (the standard code library for FRC) with tools that we found useful in our programs. We designed the library both as a way of maintaining our accumulated programming knowledge from year to year, and as a way of sharing that knowledge with other teams. Here are some of our favorite feature from Ohm:

Advanced motion control:

Complex algorithms such as PID control, motion profiling, and path following make a capable robot into an unstoppable one. Despite the invaluable advantage they offer, many teams fail to use them because of how difficult they are to implement. Ohm offers simplified interfaces that handle all the advanced math, leaving the user to simply plug in the unique details of their robot and application.

Debugging/Data logging

Every object and controller in Ohm can be asked to display all relevant information about itself for debugging. This ability hastens the debugging process and simplifies the robot code. For example, we were able to diagnose a problem with the current draw of the drivetrain in mere minutes by adding three words to the code, and watching the debugging output. Furthermore, any data that gets displayed to the dashboard can easily be saved to a log for later examination.

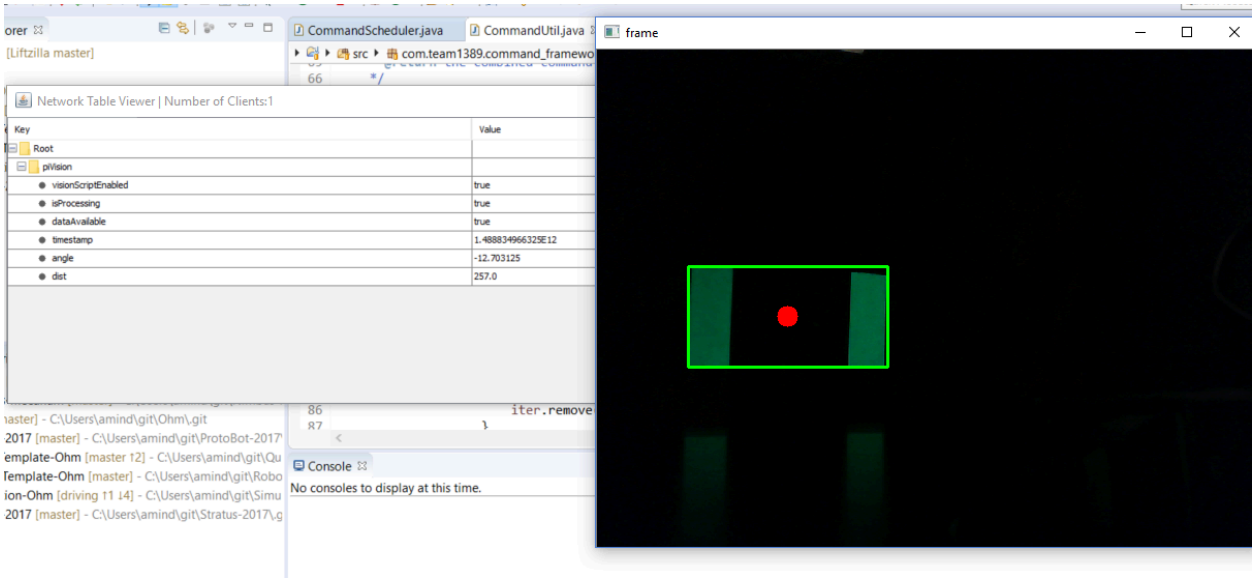
Computer Vision

Computer Vision (CV) has become critical component of high level FRC success. CV in a general sense is the use of sensors to understand your environment.

This year, our game strategy included placing as many gears as possible, so we chose to use CV to accurately and quickly align our robot with the gear peg. While CV provides unparalleled accuracy in robot operations, a working CV system must be well-planned, efficient, and reliable.

The first step of computer vision is acquiring some information about the robot's environment. In our case, a camera onboard the robot constantly captures images of the field as the robot drives, searching for the target with unique visual properties that surrounds the gear peg. We developed a processing algorithm which goes over each image pixel by pixel searching for patterns. The target produces a unique pattern when processed. Once this algorithm has identified the target, we do some trigonometry calculations to figure out the peg's position relative to the robot. Finally, we can calculate the direction the robot needs to move to get aligned.

This entire process must occur nearly instantly at the press of a button, so we optimized the entire vision system for speed. Rather than running on the main robot computer, the processing algorithm (where each pixel is analyzed) runs on a separate computer, known as a coprocessor, that can devote all of its power to vision calculations.



Autonomous Path Following

Having a successful autonomous is particularly crucial this year as the amount of points awarded for autonomous functionality are high and getting multiple rotors running is essential for success.

This year we are using a path following system that generates the most optimal path given points pre programmed into the robot. We determined the points using our simulator to find the most efficient points. We use different sets of points depending on which autonomous we would like to use. We have pre programmed eight different autonomous modes, that each utilize the path following system. The path following system generates the most efficient path possible, using, velocity, acceleration, jerk, and angle of the robot.

In autonomous every second is critical, so having the most efficient path is vital. But to optimize the speed even more, we have chosen to have the paths read off of a file rather than generated each time on the robot. This allows us to have no latency, while still having the robot follow an extremely complex path.

The utilization of the path following system has allowed us to perfect other areas of autonomous, while still having the most efficient path.

Web Dashboard

As each year's robots have become more and more complex, well-designed dashboards have become increasingly important to give drivers everything they need for peak performance. Team 1389's Web Dashboard is designed to provide the driver with information on all the most vital systems, as well as the status of the robot, all contained in a sleek and minimalistic shell. Every element of the UI overlays a video stream, which can switch between two cameras on the robot at the switch of a button. The Web Dashboard was designed using HTML, CSS, and Javascript, to allow for greater freedom in design.

System Information

The Web Dashboard keeps clutter to a minimum when it comes to systems, using transparent backgrounds and well-spaced images to allow the driver to focus on the competition. There are four widgets, representing the speed of the robot, the state of the Gear Intake, the angle of the robot, and the drive mode of the robot. Each of these systems are represented by clean, non-distracting images. All four systems are

represented together in the bottom of the screen, allowing the driver to view them with minimal eye movement from the video stream.

Robot Status

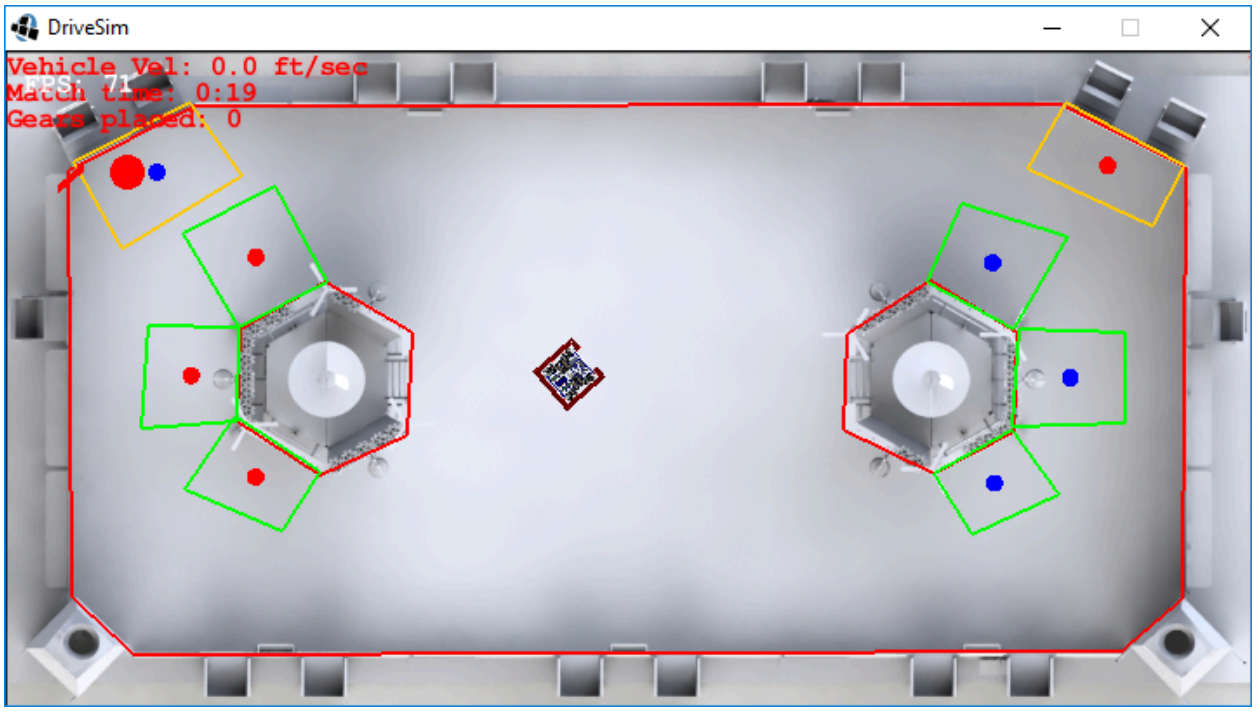
The Web Dashboard allows drivers to monitor the status of the robot, allowing for lightning fast decision making in critical situations. The amount of time left in a match can be integral as a factor for decisions, and as such is placed at the top of the screen for easy access. There are five status lights placed in the top-left corner, representing connection, drive-train, climber, ball-intake, and gear-intake. Each of these lights turn red if the system fails, and green if the system is working. In addition, a bar graph represents the current draw of each drive-train motor, giving drivers accurate knowledge on the state of each motor.

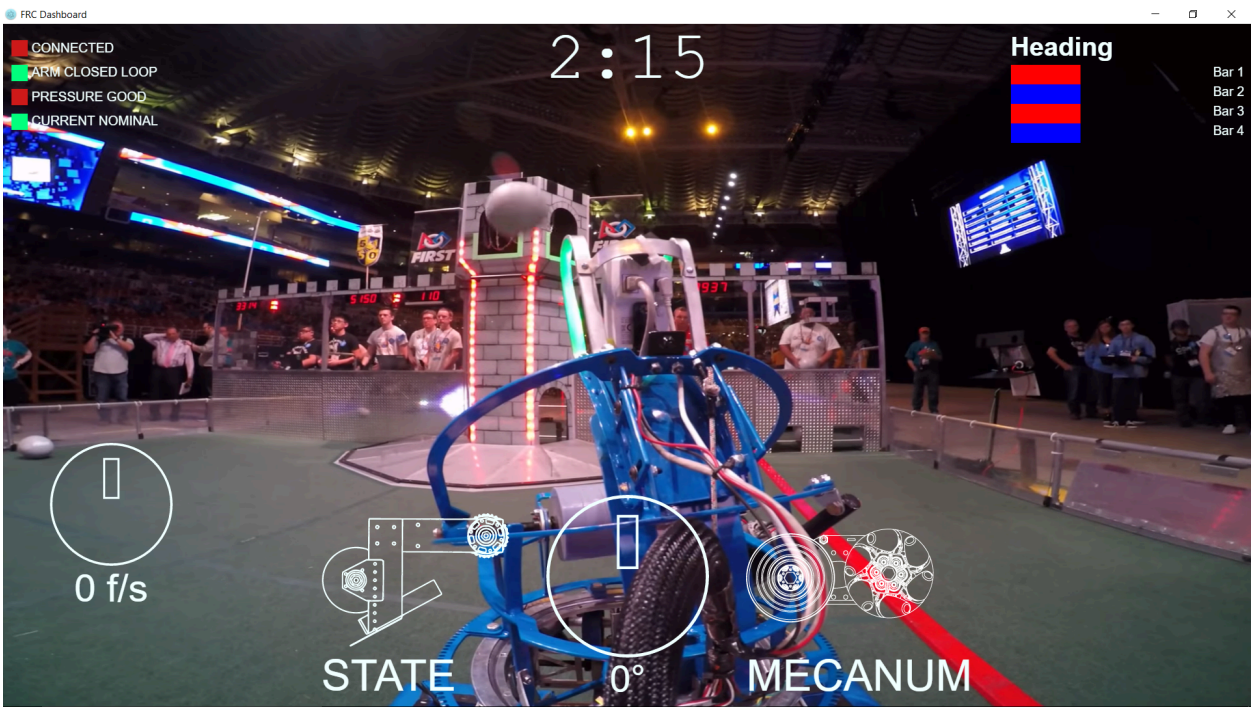
Simulation

Nothing is worse than having code ready to test but nothing to test it on. Especially in robotics, the process of testing code even when the robot is present can take a while; each time the code is changed it must be built again. The simulator is our answer to this problem. Most obviously, it allows us to practice driving the robot around the field. The robot simulation is extremely detailed. It is not merely an x and a y controlled with a joystick with an arbitrary acceleration. Each motor is simulated carefully, with an input voltage and output change in x. We then include factors like the distance between the wheels and the axles to accurately simulate the position of the robot. As we used an octo mecanum drive this year, our simulator could also switch between different modes of simulation: tank and mecanum. Finally, we also built a robust collision system that stopped the robot when it ran into a field object like the airship. The encoder would still increase but the actual position of the robot on the screen would not move. We also simulated gear pickups and gear dropoffs using a similar system. All of this gave us an easy and robust way to test both autonomous routes and teleop cycles. We even added support for joysticks and keyboard controls. We will also be able to reuse it in the future for further competitions.

However, driving is not the only thing our simulator was good for. Theoretically, anything can be simulated with our package. Through the simulator, we tested many features of Ohm, like watchers and streams. It was like having another robot to test with, except as many people could work on it as once as we wanted (with the nice fork ability of git). One interesting application of the simulator was a state estimator we built in a separate project. We simulated a network table server in the simulator with a local ip address, and connected to it with a client from the other project. Then, we could

practice displaying the robot on the driver station from only x, y, and theta values from where the robot *thought* it was. The chief reason we did this was to practice error correcting this display: the robot's position estimation values get off when the encoders get off, which happens when the robot's wheels slip. This can include hitting a wall, another robot, or just inevitable skidding. To combat this, we decided to reset the robot position client side whenever we dropped off a gear, as the angle stays relatively error free and the gear drop off point can be completely determined from the robot's angle. The driver can also click anywhere on the screen to tell the client side simulator that right now, the robot is *there*.





Date: October 26, 2024
To: Dr. Lei Huang
From: Arye Mindell
Subject: Sequential Logic Design

The objective of this experiment is to design and implement a three-bit synchronous up/down prime number counter with active-high enable. [1]

The counter cycles continuously at a rate specified by an external clock signal, with the count direction determined by a single input bit X_1, and the device enable controlled by a second input X_0. Outputs Y_0, Y_1, and Y_2 represent the 3-bit count. The counter's state is stored in two JK Flip-Flops, Q_1 and Q_0. [1]

The design implements a Moore model sequential circuit, as shown in the block diagram in Figure 1.

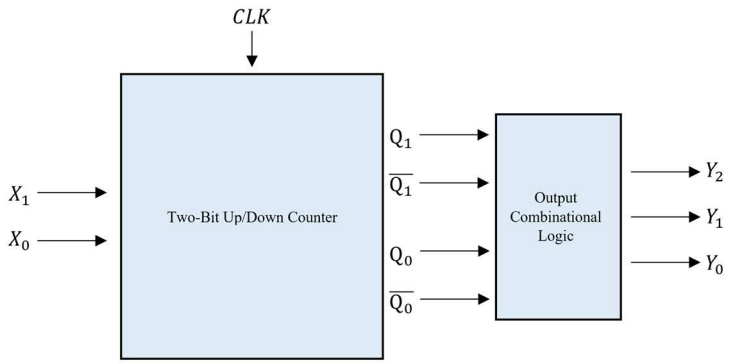


Figure 1: Prime-Counter Block Diagram

The circuit is separated into two modules: The first module, a two-bit up/down counter, manages state transitions based on the clock and input signals. The second module translates the current state into a three-bit binary representation of the corresponding prime number.

Figure 2 shows the specified state transitions and corresponding outputs. Each state represents a unique prime, as indicated by the three output bits.

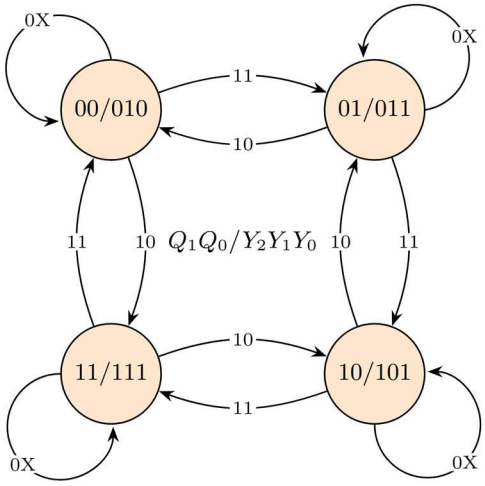


Figure 2: Prime-Counter State Diagram

Table 1(a) displays the output and next state as a function of the current state and inputs. Since the circuit employs a Moore model, the outputs are independent of the inputs, depending only on the current state.

Table 1: Prime Counter Logic Tables. (a) State Transitions and Outputs. (b) JK Flip-flop Excitation Behavior.

		X_1X_0				
State	Q_1Q_0	00	01	11	10	$Y_2Y_1Y_0$
2	00	00	11	01	00	010
3	01	01	00	10	01	011
7	11	11	10	00	11	101
5	10	10	01	11	10	111

(a)

Q	Q'	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

(b)

The circuit's state is stored in two JK flip-flops, with excitation behavior outlined in Table 1(b). Applying this behavior with the state transitions in Table 1(a) derives the K-maps in Figure 3.

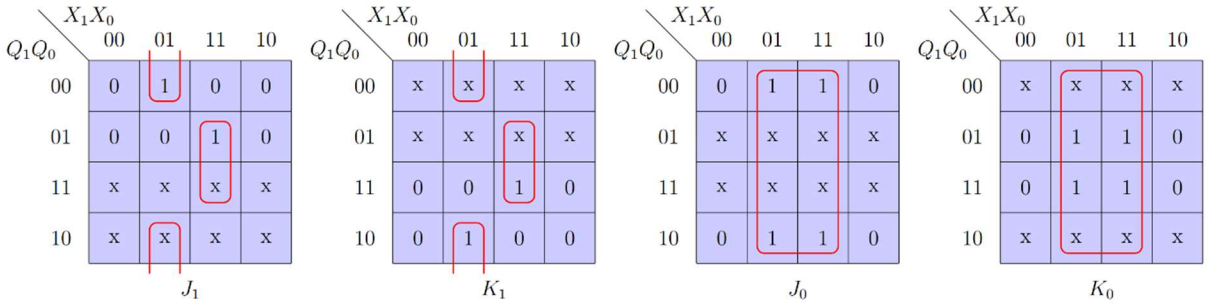


Figure 3: Flip-flop Excitation K-maps

Minimizing the K-maps in Figure 3 yields (1) and (2) which represent the excitation logic for each flip-flop input.

$$J_1 = K_1 = \left(\overline{Q_0}\,\overline{X_1}\,X_0\right) + (Q_0X_0X_1) \tag{1}$$

$$J_0 = K_0 = X_0 \tag{2}$$

Similarly, the output logic is determined from the K-maps of each output bit as a function of Q_1 and Q_0, as shown in Figure 4.

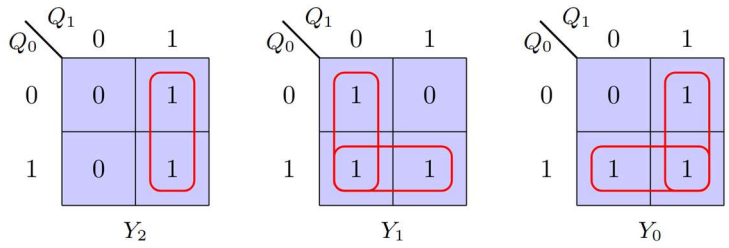


Figure 4: Output Logic K-maps

Minimizing the K-maps in Figure 4 yields (3) – (5) which describe the combinational logic of each output bit.

$$Y_2 = Q_0 \tag{3}$$

$$Y_1 = Q_0 + \overline{Q_1} \tag{4}$$

$$Y_0 = Q_0 + Q_1 \tag{5}$$

Figure 6 shows the simulation results which verify (1) – (5). The design and testbench code for this simulation are found in Attachment 1.

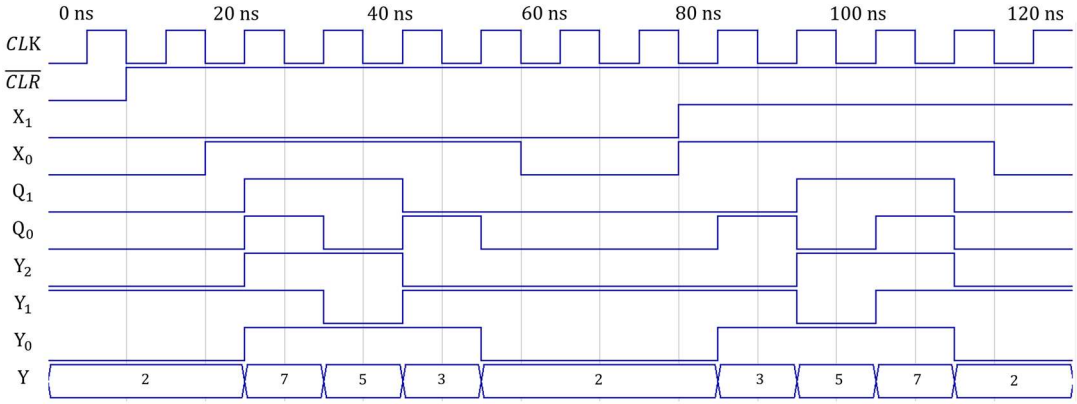


Figure 6: Prime-Counter Timing Diagram from Simulation

From 20 ns – 60 ns, X_0 is pulled high with X_1 low, which causes the output to decrement with each rising edge of the clock. From 60 ns – 80 ns both inputs are low, and the output remains unchanged. From 80-100ns, both inputs are pulled high, which causes the output to increment with each cycle. Finally, at 120 ns, X_0 is pulled low while X_1 remains high, disabling the count.

Shown in Figure 5, the final circuit design realizes the derived logic equations in modules as represented in Figure 1.

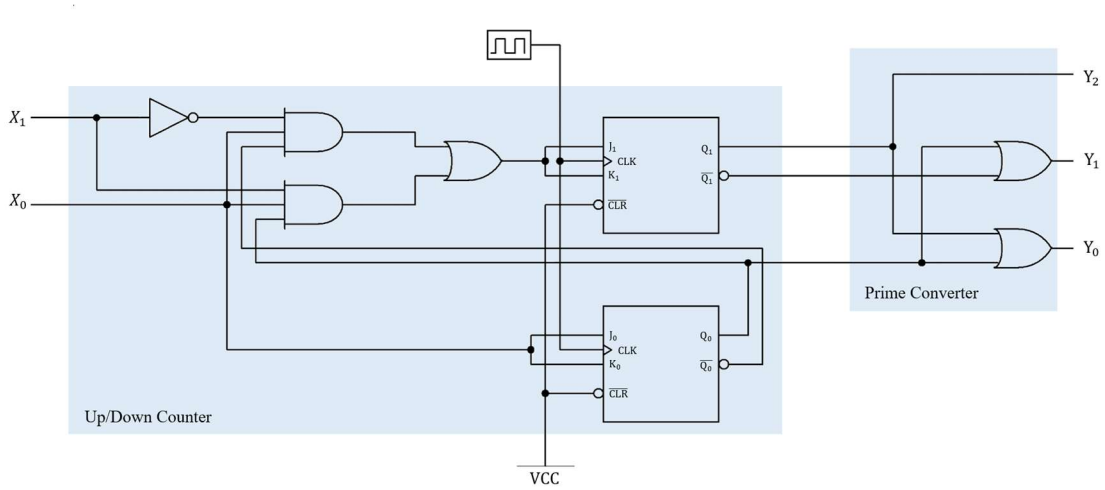


Figure 5: Prime-Counter Logic Diagram

The circuit is composed of 74LS32 OR gates, a 74LS04 Inverter, 74LS11 3-Input AND gates, and 7473 JK flip-flops. A built-in function generator inside the Tektronix 5-Series Mixed Signal Oscilloscope's drives the clock signal with a 1 Hz square wave. [1] The logic gates operate with a logic-high level of 5V, driven by a DC power supply. [2] The active-low CLR input of each flip-flop is pulled high.

Shown in Table 4, the procedure involves assembling and testing the modules sequentially. After each module is assembled, all cases in Table 1(a) are verified by pulling the inputs high or low as specified by the table and comparing the outputs indicated by LEDs to the expected output given by Figure 6.

Table 4: Experimental Procedure

Step	Action	Purpose	Materials	Expected Values
1	Assemble the Up/Down Counter module as pictured in Figure 5	Assemble Up/Down Counter	Logic Gates JK flip-flops Oscilloscope	
2	Connect LEDs at Q1 and Q0, verify proper state-cycling with each possible input combination	Test Up/Down Counter	LEDs Power Supply	See Figure 6, Q_0, Q_1
3	Assemble and connect Prime Converter module as pictured in Figure 5	Assemble Prime Converter	Logic Gates	
4	Connect LEDs at each output, verify output with each possible input combination	Test Design	LEDs Power Supply	See Figure 6, Y_2, Y_1, Y_0

Successful operation is confirmed if the LEDs display the expected sequence of prime numbers according to the input conditions, demonstrating accurate functionality of both the counter and prime conversion logic.

References:

[1] L. Huang, "Sequential Logic Design," *Memo to EECE 3100 Students*, Loyola Marymount University, Los Angeles, CA, Oct. 22, 2024. [Online]. Available: https://brightspace.lmu.edu/d2l/le/content/253458/viewContent/3109519/View.

[2] Texas Instruments, "SN74LS04 Hex Inverter," Datasheet, [Online]. Available: https://www.ti.com/lit/ds/symlink/sn74ls04.pdf?ts=1729977523913

Attachments:

1. VHDL Simulation Test Code

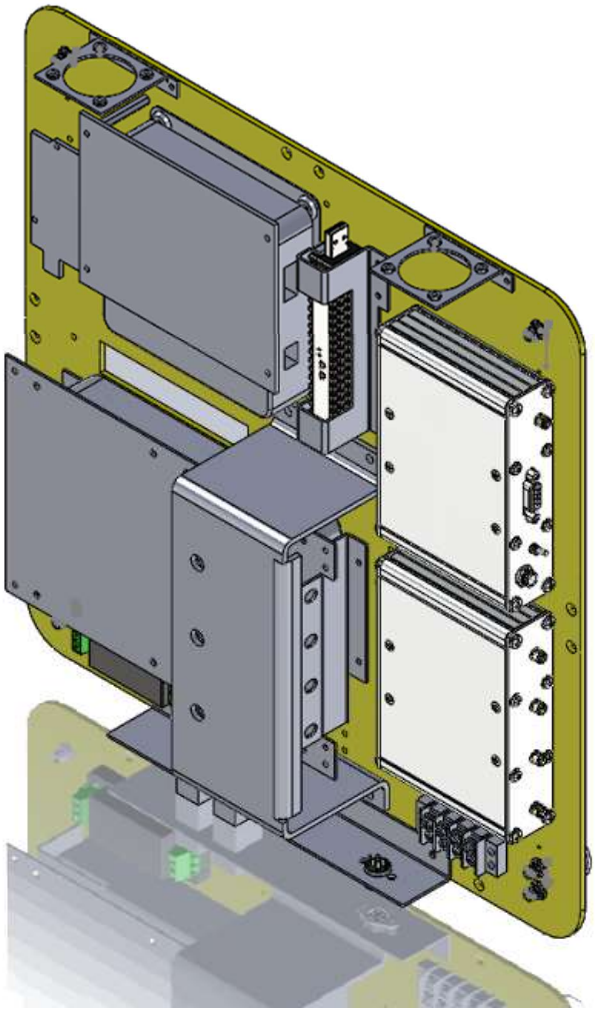
PrimeCounter.vhd

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity PrimeCounter is
    Port (
        clk : in STD_LOGIC;
        reset : in STD_LOGIC;
        X1 : in STD_LOGIC;  -- Up/Down control
        X0 : in STD_LOGIC;  -- Enable
        Y2, Y1, Y0 : out STD_LOGIC  -- Output 3-bit prime representation
    );
end PrimeCounter;

architecture Behavioral of PrimeCounter is
    signal Q1, Q0 : STD_LOGIC := '0';  -- Flip-flop states

    -- Define internal signals for JK flip-flop inputs
    signal J1, K1, J0, K0 : STD_LOGIC;
begin

    -- Excitation Logic
    J1 <= (not Q0 and not X1 and X0) or (Q0 and X0 and X1);
    K1 <= (not Q0 and not X1 and X0) or (Q0 and X0 and X1);
    J0 <= X0;
    K0 <= X0;

    -- JK Flip-Flop Process
    process (clk, reset, Q0, Q1)
    begin

        if reset = '0' then
            Q1 <= '0';
            Q0 <= '0';
        elsif rising_edge(clk) then
            -- Q1 JK flip-flop
            if J1 = '1' and K1 = '0' then
                Q1 <= '1';
            elsif J1 = '0' and K1 = '1' then
                Q1 <= '0';
            elsif J1 = '1' and K1 = '1' then
                Q1 <= not Q1;
            end if;

            -- Q0 JK flip-flop
            if J0 = '1' and K0 = '0' then
                Q0 <= '1';
            elsif J0 = '0' and K0 = '1' then
                Q0 <= '0';
            elsif J0 = '1' and K0 = '1' then
                Q0 <= not Q0;
            end if;
        end if;
    Y2 <= Q1;           -- Most Significant Bit
    Y1 <= not Q1 or Q0;     -- Middle Bit
    Y0 <= Q0 or Q1;           -- Least Significant Bit
    end process;
process(Q1, Q0)
begin

end process;
    -- Output Logic for 3-bit prime representation

end Behavioral;
```

PrimeCounter_tb.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity PrimeCounter_tb is
end PrimeCounter_tb;
architecture Behavioral of PrimeCounter_tb is
    -- Component Declaration for the Unit Under Test (UUT)
    component PrimeCounter
        Port (
            clk : in STD_LOGIC;
            reset : in STD_LOGIC;
            X1 : in STD_LOGIC;
            X0 : in STD_LOGIC;
            Y2, Y1, Y0 : out STD_LOGIC
        );
    end component;
    -- Inputs
    signal clk : STD_LOGIC := '0';
    signal reset : STD_LOGIC := '0';
    signal X1 : STD_LOGIC := '0';  -- Up/Down control
    signal X0 : STD_LOGIC := '0';  -- Enable
    -- Outputs
    signal Y2, Y1, Y0 : STD_LOGIC;
    signal Y : STD_LOGIC_VECTOR(2 downto 0);  -- 3-bit vector for binary output
    -- Clock period
    constant clk_period : time := 10 ns;
begin
    -- Instantiate the Unit Under Test (UUT)
    uut: PrimeCounter
        Port map (
            clk => clk,
            reset => reset,
            X1 => X1,
            X0 => X0,
            Y2 => Y2,
            Y1 => Y1,
            Y0 => Y0
        );
    Y <= Y2 & Y1 & Y0;
    -- Clock process definitions
    clk_process :process
    begin
        clk <= '0';
        wait for clk_period/2;
        clk <= '1';
        wait for clk_period/2;
    end process;
    -- Testbench process to test different cases
stim_proc: process
begin
    -- Reset
    reset <= '0';
    wait for clk_period;
    reset <= '1';
    wait for clk_period;
    -- Enable and Count Up
    X0 <= '1'; -- Enable
    X1 <= '0'; -- Count up
    wait for 4*clk_period;
    -- Disable
    X0 <= '0'; -- Disable
    wait for 2*clk_period;
    -- Enable and Count Down
    X0 <= '1'; -- Enable
    X1 <= '1'; -- Count down
    wait for 4*clk_period;
    -- Disable
    X0 <= '0'; -- Disable
    wait for 2*clk_period;
    -- End Simulation
    wait;
end process;

end Behavioral;
```

HUMATICS

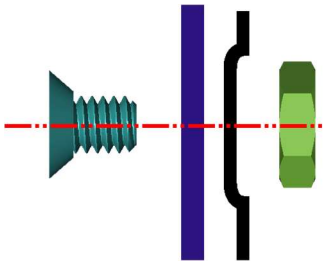


APOLLO ASSEMBLY INSTRUCTIONS

V2.1.1

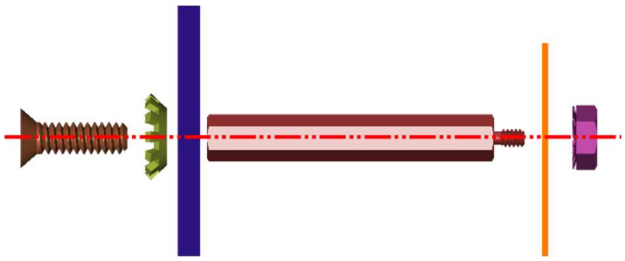
Arye Mindell

NUC Mounting Plate



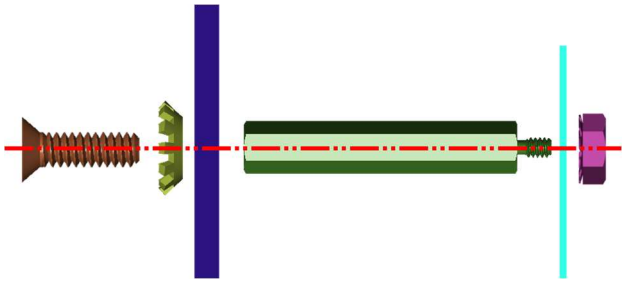
- Steel Phillips Flat Head Screws, 82 Degree Countersink Angle, 8-32 Thread Size, 1/4" Long - 90273A190
- Interrogator Plate
- NUC-VESA Mount
- Zinc-Plated Steel Narrow Hex Nut, 8-32 Thread Size - 90760A009

HPB Bracket



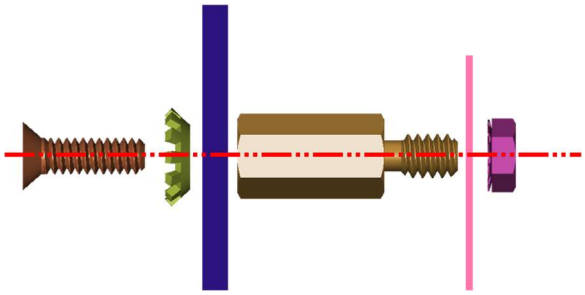
- Steel Phillips Flat Head Screw, 82 Degree Countersink Angle, 4-40 Thread Size, 3/8" Long - 90273A108
- Countersunk External-Tooth Lock Washer, Zinc-Plated Steel, Number 4 Screw Size, 0.113" ID, 0.213" OD - 90069A103
- Interrogator Plate
- 12L14 Steel Male-Female Threaded Hex Standoff, Zinc-Plated, 1/4" Hex Size, 1-3/4" Length, 4-40 Thread Size - 93620A001
- HPB bracket
- Steel Locknut with External-Tooth Lock Washer, Zinc-Plated, 4-40 Thread Size - 90675A005

Pluto Bracket



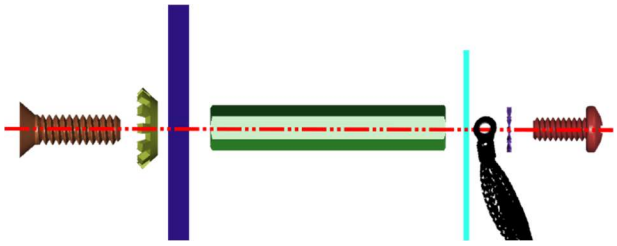
- Steel Phillips Flat Head Screw, 82 Degree Countersink Angle, 4-40 Thread Size, 3/8" Long - 90273A108
- Countersunk External-Tooth Lock Washer, Zinc-Plated Steel, Number 4 Screw Size, 0.113" ID, 0.213" OD - 90069A103
- Interrogator Plate
- 12L14 Steel Male-Female Threaded Hex Standoff, Zinc-Plated, 1/4" Hex Size, 1-1/2" Length, 4-40 Thread Size - 93620A438
- Pluto bracket
- Steel Locknut with External-Tooth Lock Washer, Zinc-Plated, 4-40 Thread Size - 90675A005

PLL Bracket



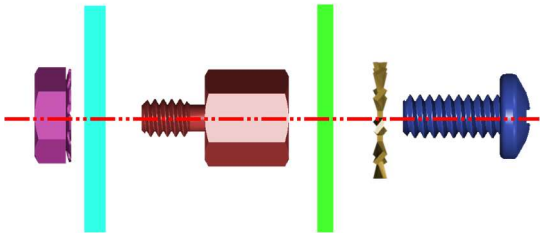
- Steel Phillips Flat Head Screw, 82 Degree Countersink Angle, 4-40 Thread Size, 3/8" Long - 90273A108
- Countersunk External-Tooth Lock Washer, Zinc-Plated Steel, Number 4 Screw Size, 0.113" ID, 0.213" OD - 90069A103
- Interrogator Plate
- 12L14 Steel Male-Female Threaded Hex Standoff, Zinc-Plated, 1/4" Hex Size, 3/8" Length, 4-40 Thread Size - 93620A431
- PLL bracket
- Steel Locknut with External-Tooth Lock Washer, Zinc-Plated, 4-40 Thread Size - 90675A005

Pluto Bracket (Grounding Strap)



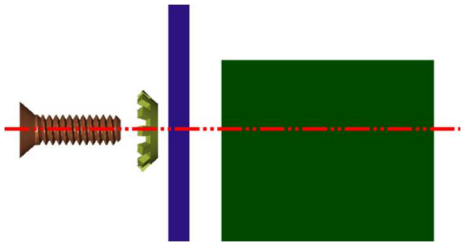
- Steel Phillips Flat Head Screw, 82 Degree Countersink Angle, 4-40 Thread Size, 3/8" Long - 90273A108
- Countersunk External-Tooth Lock Washer, Zinc-Plated Steel, Number 4 Screw Size, 0.113" ID, 0.213" OD - 90069A103
- Interrogator Plate
- Zinc-Plated Steel Female Threaded Hex Standoff, 1/4" Hex Size, 1-1/2" Length, 4-40 Thread Size - 91920A171
- Pluto bracket
- Grounding Strap
- Zinc-Plated Steel External-Tooth Lock Washer for Number 4 Screw Size, 0.115" ID, 0.26" OD - 91114A005
- Steel Phillips Rounded Head Screws, 4-40 Thread Size, 5/16" Long - 90272A107

Pluto Mounting



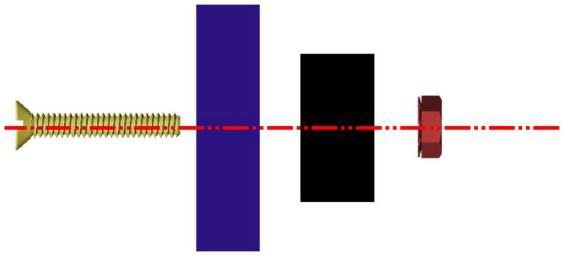
- Steel Locknut with External-Tooth Lock Washer, Zinc-Plated, 4-40 Thread Size - 90675A005
- Pluto Bracket
- 12L14 Steel Male-Female Threaded Hex Standoff, Zinc-Plated, 1/4" Hex Size, 1/4" Length, 4-40 Thread Size - 93620A430
- Pluto Board
- Zinc-Plated Steel External-Tooth Lock Washer for Number 4 Screw Size, 0.115" ID, 0.26" OD - 91114A005
- Steel Phillips Rounded Head Screw, 4-40 Thread Size, 1/4" Long - 90272A106

Rx/Tx Module



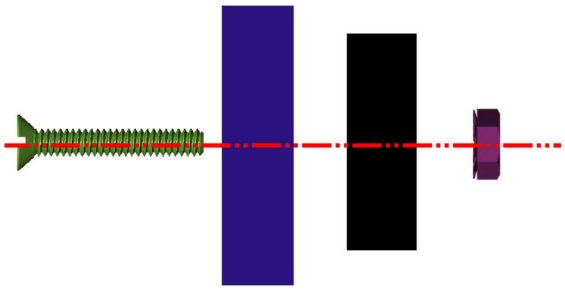
- Steel Phillips Flat Head Screws, 82 Degree Countersink Angle, 6-32 Thread Size, 5/16" Long - 90273A145
- Countersunk External-Tooth Lock Washer, Zinc-Plated Steel, Number 6 Screw Size, 0.14" ID, 0.289" OD - 90069A105
- Interrogator Plate
- Rx/Tx Module

DC Distribution Block



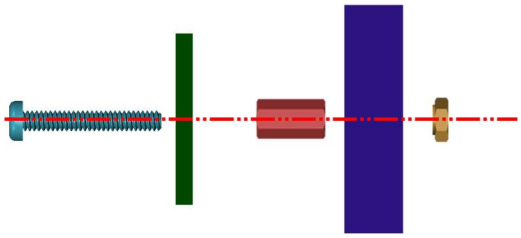
- Steel Phillips Flat Head Screws, 82 Degree Countersink Angle, 6-32 Thread Size, 3/4" Long - 90273A151
- Interrogator Plate
- DC Distribution Block
- Zinc-Plated Steel Locknut with External-Tooth Lock Washer, 6-32 Thread Size, 1/4" Wide - 90413A101

DC Converter



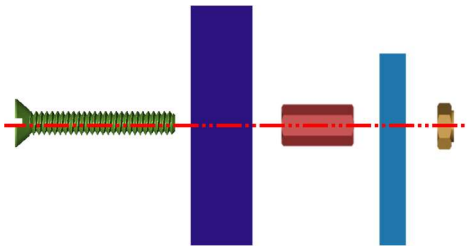
- Steel Phillips Flat Head Screws, 82 Degree Countersink Angle, 4-40 Thread Size, 3/4" Long - 90273A113
- Interrogator Plate
- DC Converter
- Steel Locknut with External-Tooth Lock Washer, Zinc-Plated, 4-40 Thread Size - 90675A005

LED Board



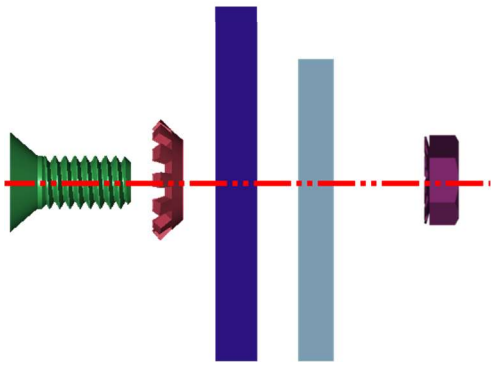
- Off-White Nylon Rounded Head Screws, Phillips Drive Style, 4-40 Thread Size, 3/4" Long - 93135A278
- LED Board
- Female Threaded Hex Standoff, 6/6 Nylon, 3/16" Hex Size, 3/8" Length, 4-40 Thread Size - 92319A869
- Interrogator Plate
- Nylon Hex Nut, 4-40 Thread Size - 94812A200

Arduino



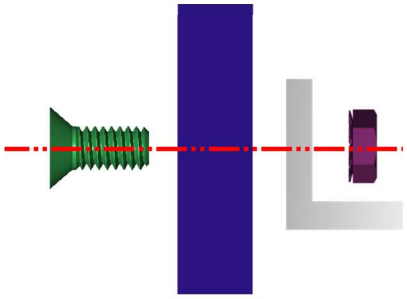
- LED Board
- Interrogator Plate
- Female Threaded Hex Standoff, 6/6 Nylon, 3/16" Hex Size, 3/8" Length, 4-40 Thread Size - 92319A869
- Off-White Nylon Slotted Flat Head Screws, 82 Degree Countersink Angle, 4-40 Thread Size, 3/4" Long - 94605A113

Mounting Bracket



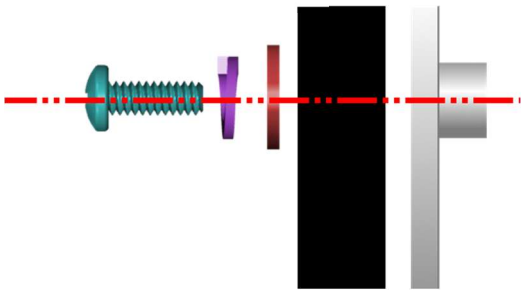
- Steel Phillips Flat Head Screws, 82 Degree Countersink Angle, 8-32 Thread Size, 3/8" Long - 90273A192
- Countersunk External-Tooth Lock Washer, Zinc-Plated Steel, Number 8 Screw Size, 0.167" ID, 0.322" OD - 90069A110
- Interrogator Plate
- Mounting bracket
- Steel Locknut with External-Tooth Lock Washer, Zinc-Plated, 8-32 Thread Size, 11/32" Wide - 90675A009

Fan Bracket



- Steel Phillips Flat Head Screw, 82 Degree Countersink Angle, 4-40 Thread Size, 3/8" Long - 90273A108
- Interrogator Plate
- Mounting bracket
- Steel Locknut with External-Tooth Lock Washer, Zinc-Plated, 4-40 Thread Size - 90675A005

Fan Mounting



- Steel Phillips Rounded Head Screws, 4-40 Thread Size, 5/8" Long - 90272A112
- Zinc-Plated Steel Split Lock Washer for Number 4 Screw Size, 0.12" ID, 0.209" OD - 91102A720
- Zinc Yellow-Chromate Plated Grade 8 Steel Washer for Number 4 Screw Size, 0.125" ID, 0.312" OD - 98023A111
- Fan
- Mounting bracket

Winter 2016 Programming Contest

Team: The_Flatlanders

School: Walt Whitman High School

Students: Joshua Engels, Rory Nevins, Joey Wood, Ari Mindell

Code Related Aspects

Correctness/Completeness

- Does it work and how well?
- Have other potential uses for the code been considered?
- Does it solve the problem for varied inputs?

Your solution works very well. We did not experience any crashes or issues with your software while it was running. The output for each package (details) seems to be very complete and correct. Your solution seems to implement all of the requirements we put forth for our package tracking interface. One thing I was searching for in admin mode was the ability to see a list of all packages that are out for delivery (I think the mechanism you have in place is to essentially look at the map, and you'll see a marker for each package that you need to individually click on). This was not a requirement, but it was a common feature in other complete solutions and seemed to make admin mode more usable.

When we launched the application interface, we were ultimately opening the site from a file, rather than an address. Was this intentional, or were we opening the site incorrectly? The difference here is that the address in the url bar for the site was file:// instead of http://. Ultimately, we were not able to test multiple users hitting the site at the same time from different machines on the network, because the site was not actually deployed. All interaction with your application was done using the machine that it was installed on.

Our initial use of the application left us scratching our head because we used the UI to create our user, and did not see a way to create an admin user. The more we thought about it, this must have been a security concern on your part, because you would be giving any user that uses the interface the ability to mark their account as admin. The inclusion of the Groovy script on the server that had the ability to create both types of users certainly answers the mail to allow us to log in as either admin or end user. This sort of consideration speaks to the level of completeness that your team achieved.

Your team should be very proud of yourselves for implementing an excellent solution to the given problem. Bravo.

Modularity/Usability

- How easily can it be integrated into the code base?
- Could it be integrated into a tool chain?
- Are there logical separations between components?

One example of excellent modularity is the user account code that could be triggered as a standalone groovy script, or could be triggered through the application to create users. You guys did a good job keeping the user admin code modular enough that it could be used again in a completely different application.

One thing to consider would be moving the code that computes distance and midpoint out of the Coordinate class and into a common utilities class. Imagine that you are a company like Uber and you are growing rapidly. A whole bunch of features are being added that will also need to calculate distance. If you keep your computations locked away in individual classes, you will discourage other people from using the same algorithms/code. If you had a static utilities class to do all math or all geo computations (e.g. GeoUtilities.groovy), you would encourage all developers that are adding additional features to use a single, common solution (instead of 10 different distance algorithms, all calculating differently).

Your code is very thin. In that respect, it is very easy to work with in terms of integrating into an existing code base. Nice job avoiding code bloat.

Maintainability/Documentation

- Were the comments sufficient and variables appropriately named?
- Could the code be understood/fixed/extended easily?

It is always appreciated when people add javadoc style comments to explain the expectations for methods. You guys did this for a lot of methods but not all of them. Since so many of them were done, it might have been cool to generate actual javadoc documentation or the Groovy equivalent (groovdoc?).

There were a handful of magic numbers found in your code (e.g. 3600 – could have been saved as a static final variable called SECONDS_PER_HOUR). Otherwise, you guys did a great job self-documenting your code with descriptive method and variable names. All of the calculation logic (midpoint, getDistance) was easy to follow and cleanly coded. The actual code blocks contained minimal comments, but the code was clean and concise enough to probably not need much at all.

I certainly felt that I could take your project and extend it (add new functionality, or update existing functionality, or borrow aspects for other projects). Code is easy to understand.

Performance/Scalability

- Does the applications performance meet explicit/implicit needs?
- Do the algorithms scale appropriately in time/memory?

From a scalability perspective, I thought it was interesting that you modeled a coordinate point as a class because each package potentially has a tremendous amount of coordinate points, depending on how long it is out for delivery. I recognize that you used the coordinate point class as a data container to hold the coordinates, as well as a location for the logic associated with computing distance and midpoint, but that could be a very large number of objects going onto the heap and you could potentially blow out your memory just from holding onto coordinate data (since each package holds an array list of all of the past points). It might make more sense to push everything out to a database so that you don't blow out memory as you scale up.

No problem was observed with the performance of your application as we sent a handful of packages through. Everything within the application remains responsive.

User Experience Aspects

Delivery

- How well were the submitted components labeled?
- Were the documentation/source/executables logically separated?

Your team organized your delivery well in terms of logically separating components. All of the web viewable code was in the htdocs directory, all of the package sending code was in Event Simulator and all of the Groovy code for your solution was in the src directory. Thanks for putting the documentation at the top level instead of burying it deep in the delivery.

That was thoughtful to include the GRE so that the user did not need to download Groovy in order to send packages (using your .bat files). It was interesting that the GRE was not pulled up to a higher level in the delivery and used to run the Groovy code in src. I suppose that this was influenced by the fact that you wanted the user to launch the application through Eclipse (and to have installed Groovy support for Eclipse), but there was still a disconnect between the delivered GRE and the fact that the user had to go install Groovy through Eclipse.

Documentation

- Was the supplied documentation easy to understand?
- Was the acceptable input format well defined?
- Was the tools output well explained?

Your team provided good documentation in the README.md file, separated into Product Documentation, Software Documentation, and Software Requirements. A text based document is

always a good choice because it doesn't assume that the user has Microsoft Office or Adobe Reader installed. Thank you for including Eclipse plug-in installation paths in your documentation. We definitely had to lean on the documentation to figure out how to get the software running as well as to figure out how to make accounts.

Build Environment

- Was it easy to build and execute the application?
- Were all details of the build explained well?

The details of how to build/execute were well explained in your documentation. Build Environment was one of the areas that turned out to be a huge differentiator between all of the projects. Ultimately, the ideal solution here would be some sort of installer, or set up script that takes care of everything, or prepackaged executable containing all dependencies. All of those minimize the amount of work that a user has to do in order to get the application up and running. We did not have any problem getting your project set up, but it required significantly more effort than some of the other projects.

User Interface

- Was the interface easy to interact with?
- Did the solution execute easily?
- Was all presented information understandable?

The user interface for your application certainly contained everything that we were looking for, particularly the integration with a mapping api to display location of packages. You guys had some features that were really nice, such as tracing the route on the maps and custom notes for packages.

One thing that was a tiny bit clunky was the Refresh Map button. It seemed like that always reset the zoom to the world level. We noticed that when we were zoomed in on a package, the package did not 'move'. To see the movement of a package, it seemed that you had to refresh the map, then zoom back into the package.

Another aspect that was slightly clunky, specifically in Admin mode, was a lack of a list of all of the packages. It seems like the only way to access a package is by clicking on it on the map, instead of seeing a list of all packages that are out for delivery. It also seems like the only way to know how many packages are out for delivery is to count the number of markers on the map.

The details that you provided for each package were excellent. No question there.

Requested Feedback

- **We used cookies in our final solution to keep track of the currently logged in user. We knew all along that this was a particularly fragile method of storing secure data, and we did our best to optimize it by storing only the username and not the password, but we understand that regardless, a user with cookies disabled would not be able to use our solution. We ended up running out of time to resolve this issue, and instead put a message at the bottom of the login screen explaining that cookies needed to be enabled. Is this solution acceptable? if not, what would have been a better way to implement it?**

It is acceptable inasmuch as it is a rapidly developed solution that works and it demonstrates the concept. Cookies would not be my preferred mechanism for a publically deployed solution on the internet, because of security concerns (cookies can be manipulated on the client side). Most back ends use sessions to keep track of the unique users interacting with the site.

- **Another bug that remained in our code after submission was the interaction with chrome web browser. We were unable to find a way to force chrome to allow our static HTML to send post requests to the servlet. Is there a way to do this without hosting everything on the actual internet?**

The problem that we were seeing on our end, when using a Chrome browser, was a 'null' value for the Origin header (just to make sure we are on the same page here). The nature of the problem seems to be coming from the fact that we were launching the website from a file (file:// was the address) instead of from a url (http://). The problem can be solved by running a web server locally, and placing the files in the web server's root directory. That way, you can use a url to access the files, such as http://localhost/mainPage.html.

- We realized upon reading your email that our installation process involved installing eclipse to be able to run the server. Should we have compiled the server and simply had an executable file for the idt user to run?

The final submissions that we saw for this contest ranged from development environments to full blow installation scripts. We did get your software up and running, but we were using a Windows virtual machine that did not already have eclipse (or any development environment) already installed, so it took a bit of work. I think it is awesome that you did deliver a project that we could import, look at the code, and compile so easily. An additional piece to your delivery that would have been very convenient for IDT would have been an executable version of your solution.