Final Design Report

Spring 2013 - EML4552

Florida Agricultural & Mechanical University / Florida State University
College of Engineering

Team Members:

Electrical and Computer Engineering

Ricardo Asencio

Matthew Wilson

Mechanical Engineering

Myles Bean

Daniel Bucken

Parker Harwood

Jason Rhodan

Project Advisors:

Dr. Jonathan Clark, PhD

Department of Mechanical Engineering

Dr. Rodney Roberts, PhD

Department of Electrical and Computer Engineering

Contents

Table of Figures

I. Sub-Systems Overview (NASA Mid-Point Review)

A. Abstract

This document describes the FAMU/FSU College of Engineering's rover design for the 2013 RASC-AL Robo-Ops competition. The team consists of 6 undergraduate engineering students all with an interest in space exploration and a strong will to compete in this year's competition. Guidance and working facilities were provided to the team by their main advisor, Dr. Jonathan Clark, and the STRIDe Lab, which operates under his direction.

A hexapedal locomotion platform forms the basis for the proposed rover, and grants the rover several key features which we believe will make it successful in completing the tasks of this year's competition. The rover also features a low degree of freedom Sample Extraction Module (SEM) designed specifically for the legged platform, an on-board Field Programmable Gate Array to consolidate logic operations (decoding of motor signals), and a strategy for wireless control of the rover, which minimizes on-board computing requirements.

B. Team Leads and Facilities

Daniel Bucken, Mechanical Systems Lead – Daniel Bucken is a senior at Florida State University pursuing his BS in Mechanical Engineering. Over the past year he has been employed by the Center for Intelligent Systems, Controls and Robotics (CISCOR) as a research assistant. His work has focused on the design of robotic systems and the design of components for implementation of controls on existing platforms.

Ricardo Asencio, Electrical and Computing Systems Lead – Ricardo is a senior at Florida State University and is completing his degree in Computer Engineering. He recently completed a year-long internship with Intel Corporation in Folsom, California and plans to return for full-time work after graduation. At Intel, Ricardo had various roles but was primarily focused on system validation of modern application-specific IC's while running a complete software stack in a pre/post silicon environment. His interests include autonomous robotic systems and artificial intelligence.

STRIDe Lab, Working Facilities – Scansorial and Terrestrial Robotics and Integrated Design Lab was founded in 2007 by its director, team advisor Dr. Jonathan Clark, with the aim of developing robotic platforms which can challenge the agility and versatility of animals and insects. STRIDe Lab has worked extensively on the design and control of legged platforms and is well equipped for the task of developing a legged rover. The lab boasts several tools to aid in the manufacture of a rover including a laser cutter, composite material construction tools, extensive analysis and testing devices, and a capable machine shop available next door at the FAMU/FSU College of Engineering.

C. Rover Design and Performance Summary

Mechanical		Electrical/Computing	
Stowed Dimensions:	89 cm x 72 cm x 50 cm	**On-Board Computing:**	Raspberry Pi
Weight:	44.5 kg	**Computing Power:**	700 MHz
Ground Clearance:	14 cm	**Logic Device:**	Xilinx Spartan-6 FPGA
Tipping Angle:	43.2˚	**Operating System:**	Arch Linux
Claw Movement Speed:	4 cm/s	**Control Method:**	SSH
Top Speed:	0.8 m/s	**Networking:**	Verizon 4G LTE USB modem

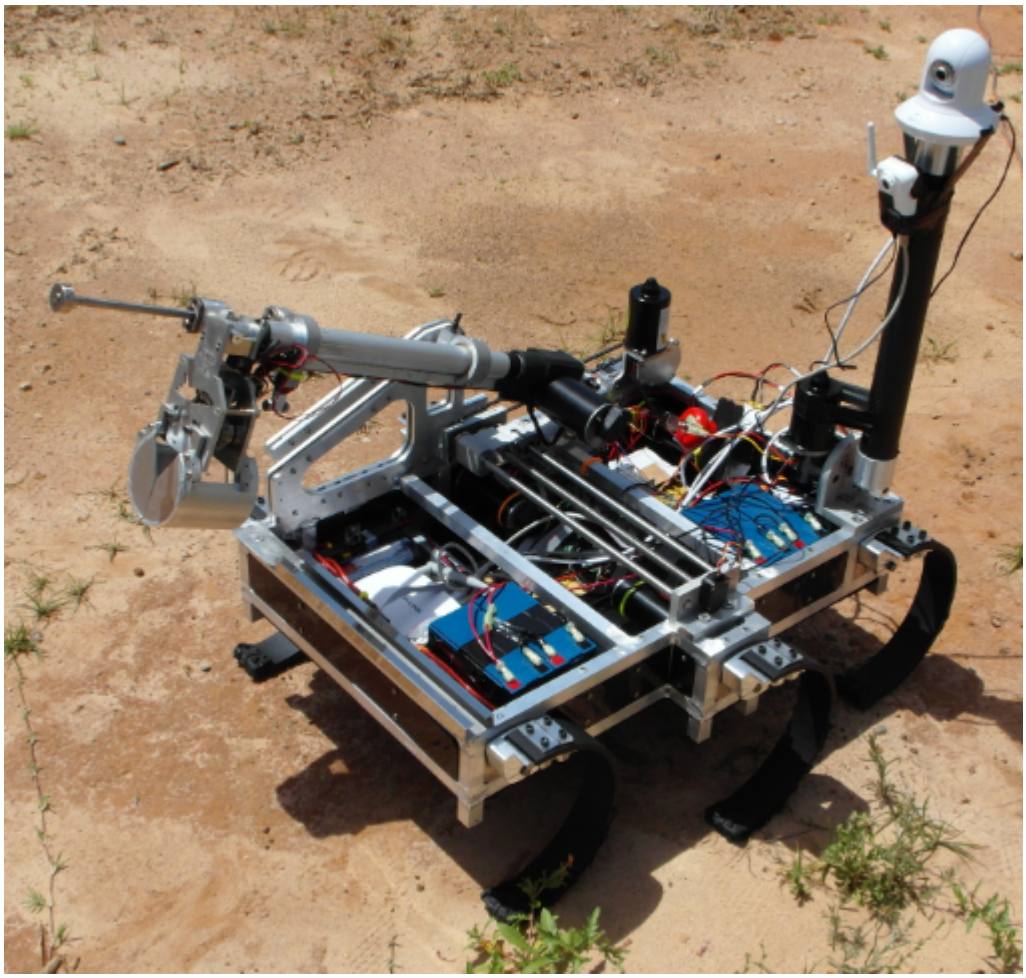


Figure 1 - An image of the rover in the standing position during testing.

D. Rover Subsystems

Computing

The computing architecture for our rover consists of three subsystems: the communication hardware, the central computing hardware, and the power delivery systems. The computing hardware consists of two main onboard computers (Raspberry Pi), and a Field-Programmable Gate Array (Xula-2). The Raspberry Pi, directly controls all other hardware subsystems present on the rover. The FPGA provides six hardware quadrature decoders for counting motor positions and six Pulse Width Modulated

(PWM) signal drivers; this hardware provides one Raspberry Pi the basis to execute the motor control algorithm. The FPGA pins connect to six high-current motor controllers that provide the high current/voltage required to move the leg motors.

Communications System

The rover's primary means of communication with Mission Control is through a Verizon Wireless 4G USB modem. The modem is connected to a TP-Link router which provides internet connectivity to the main onboard computers and the Internet Protocol (IP) cameras. The communication system allows for Secure Shell Handler (SSH) access to the Raspberry Pi computers, thus providing control of the rover over the wireless broadband connection. One onboard computer (Raspberry Pi) communicates with the FPGA over a Serial Peripheral Interface (SPI) bus. The SEM motor drivers are connected to another Raspberry Pi via UART protocol and provide control of the motors for the gripper and arm. The IP cameras are directly connected to the router over Ethernet and are controlled remotely by the operator through the GUI.

Vision System

The primary vision system components are the IP cameras, camera boom, GUI, and blob detection algorithm. A wired, mast-mounted, pan/tilt IP camera (TP-Link TL-SC4171G 0.3 Megapixel) is the main camera used during navigation of the robot. Two additional IP cameras are fixed to the robot for use during sample extraction (TP-Link TL-SC3230 1.3 Megapixel) to provide a closer view of the extraction area and to determine the Cartesian location of the samples through the use of grid overlays. All cameras record internally to removable media and also broadcast to TP-Link's web servers so the Raspberry Pi does not handle video processing. The GUI extracts the video feeds and is able to apply blob detection and/or overlays. The Blob Detection software assists with sample extraction by highlighting colored rocks in the streaming video feed.

Drive System

Space-Hex's mobility system is based on a hexapedal design consisting of six single-actuated, compliant legs. Stable propulsion is achieved using an alternating tripod gait, and leg speed is governed by position-based speed variation called a Buehler Clock. The Buehler Clock speed variation consists of a sweeping phase (high speed) and a propelling phase (low speed), and each phase occurs in half the cycle time. Each leg motor is powered by a dedicated high-current motor driver, rated for 160 Amps continuous and 300 Amps peak current. The FPGA (mentioned above) passes the encoder values to the Raspberry Pi, which compares the actual position of each motor to an ideal calculated position. These two values are then compared and a PD algorithm is used to calculate the new duty cycle of the motor, which is converted to a PWM by the FPGA and sent to the motor to adjust its speed.

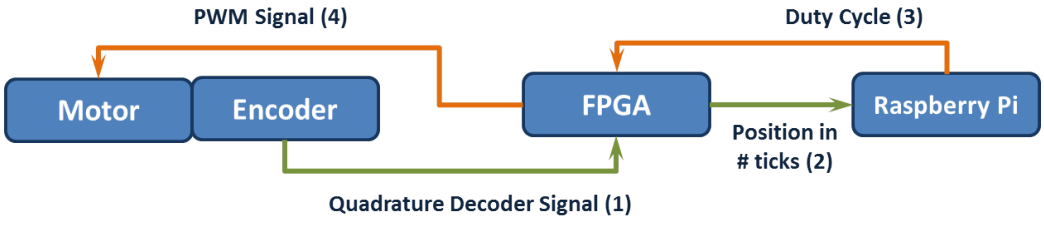


Figure 2 - Drive system diagram.

Sample Extraction Module (SEM)

A planar sample extraction system is used to complement the legged design of the rover. At the foundation of the SEM is a carriage driven by a 1 inch per revolution lead screw, which is connected to 18 Nm torque motor via a 1:1 chain drive. This

allows for motion along the width of the robot (referred to as the X-axis). Mounted to this carriage is a linear actuator with 14 inches of extension and 200 lbf of axial force which adds a second degree of freedom lengthwise along the robot (referred to as the Z-axis). Attached to the end of the linear actuator is a "Clamshell" style gripper which uses counter-rotating scoops to capture rock samples. A 4:1 chain drive and 1:1 gear set are used to increase the clamping force (5 lbf at the scoop tips) and achieve counter rotation. A cam-follower system is used to raise the gripper over the storage box on the front of the rover during sample storage

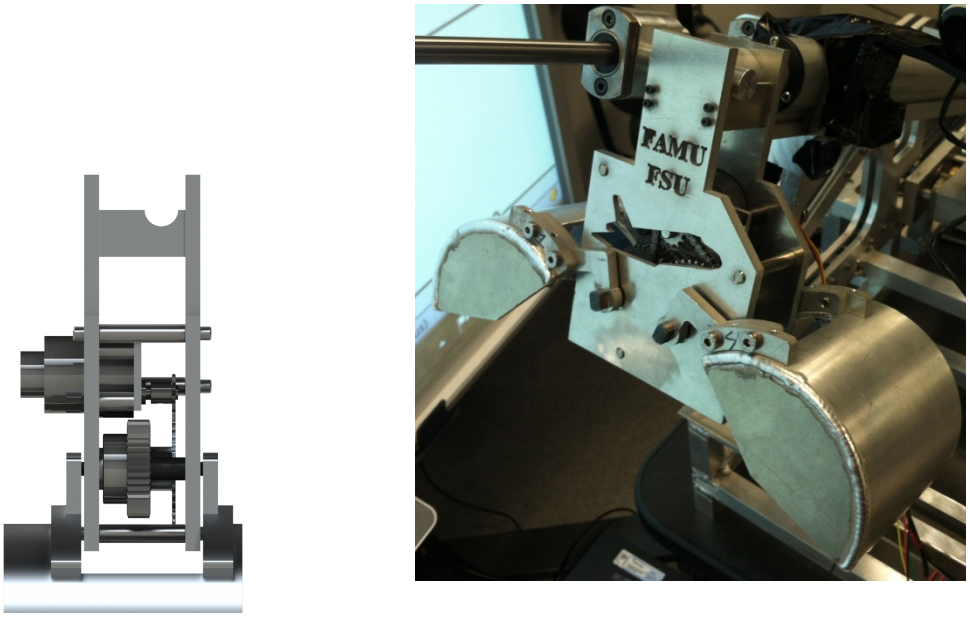


Figure 3 - Side view render depicting gripper drivetrain (left). Picture of mounted gripper in the open position (right).

Power System

The power system utilizes three batteries to power the rover's subsystems. The batteries are connected in parallel and are rated at 29.6V 50 Ah when combined. Power is supplied to all the components by these batteries using step-down switching voltage regulators where needed. A 12V regulator provides power to the communication hardware (router and modem) and IP cameras. A 5v regulator provided by the SEM motor drivers powers the Raspberry Pi computers and FPGA. The main power system component is the motor drivers.

Mission Control

Control of the rover will be performed using a Graphical User interface (GUI) that integrates rover locomotion and sample extraction controls, video display and processing, and network communications. One main operator will control the rover with another operator monitoring the sensor outputs on the rover and acting as a second set of eyes to spot samples. An approximate rover path will be determined prior to the start of the competition run that will allot a set amount of time in each area of the rock yard.

A more advanced route planning strategy is being considered in which a high resolution camera is used to scan the rock yard from the high vantage starting point of Mars hill. The route could then be planned to maximize score based on the visible rocks. More testing is needed to determine the viability of this strategy.

II. Appendices

A. Competition Rules and Requirements

Competition Summary

The mission is to develop a planetary rover platform capable of exploring the NASA Johnson Space Center (JSC) Rock Yard. At the JSC Rock Yard, the rover must acquire and store specific rock samples spread across the rock yard. The rover must be controlled from the team's home campus over a commercially available broadband connection. Teams must also connect with the general public throughout development using online social media, and community activities. Participation includes a potential grant for rover development, from NASA, worth a total of $10,000. Winning teams can receive up to $12,000 in cash prizes.

Requirements for 2012

Participation was established first by submitting a notice of intent to the competition stewards at NASA. Next, it was required that an 8 page proposal be submitted (was due December 9, 2012) to be eligible to become one of 8 teams chosen to compete. Being selected nets a team a $5,000 grant to construct the proposed rover, and all teams will be notified of their selection by December 19, 2012. The proposal was required to outline the team's plan for meeting the milestones of the competition (to be discussed later), and how the team's product will meet the following Rover Design Requirements outlined in the competition guidelines.

Rover specs for competition trim

In the rover's "Stowed configuration", meaning with all peripherals retracted, the rover must not exceed dimensions 1m x 1m x0.5m. The maximum mass (without payload) must not exceed 45kg, or else points will be deducted. No internal combustion engines are allowed, and the rover must be water-proof.

Rover performance and capability required

The rover must be capable of traversing obstacles at least 10cm tall, negotiate +/- 33% grades, and traverse level sand surfaces for at least 20 feet of distance. The areas of the JSC Rock Yard to be included in the competition are the Rock Field, Lunar craters, Sand Dunes, and the Mars Hill. The rover must selectively acquire at least five irregularly shaped rocks while traversing the JSC Rock Yard. The rocks are outlined as having diameters from 2 - 8 cm, masses from 20 - 150 gm, and be of different colors each corresponding to a point value. The rover must store and carry these rocks throughout the course. The JSC Rock Yard and the rocks of interest can be seen in Figure 4, below.

Figure 4.: Photo of JSC Rock Yard from NASA.com (left), and photo of colored rock samples from nianet.org (right).

Controls and Communications Requirement

As stated before, the rover must be remotely controlled from the team's home campus over a commercial cellular data network (ie. via wireless broadband card). Rover data must be sent from the rover itself to operators and spectators online. This data is required to consist of live video feed and some rover telemetry. The video feed must be capable of distinguishing color (rock samples), and must be recorded and posted on the team's website.

Requirements for 2013

After being selected to compete in the 2013 RASC-AL Robo-Ops competition, the team continuously documented and broadcast rover development progress throughout the semester. Two reports are required to outline how the team has met certain milestones, and the team's confidence in completing the project on time. Next, each report will be introduced along with the milestones the team is expected to cover in that report.

Mid-Project Review Report + Video – due March 15, 2013

The purpose of this report was to display to the competition stewards that a team is on schedule to completing a rover capable of satisfying all design and performance requirements. This report consisted of a five-page written portion and a YouTube video. The team's report demonstrated the rover's present functionality in the YouTube video (viewable on the team's channel: RoboOps FAMU FSU C.O.E.), and chronicled problems encountered and solutions planned. On April 9, 2013, the competition stewards at NASA approved of the team's reporting, and awarded the team an additional $5,000 grant to the team.

Outreach Video – due May 17, 2013

Each team is required to conduct educational public outreach (E/PO) via the internet and in person. The team's Facebook page (www.facebook.com/FAMUFSUHexcavator), YouTube channel, Instagram page (famu_fsu_roboops), and a team website (eng.fsu.edu/me/senior_design/2013/team11) were used to host updates on rover development, relevant documents, and video. The outreach video itself is for teams to generate interest for the team itself, science, and space exploration in general, and must be posted on the team's website. Links to the team's website, E/PO pages, and the Rover's Camera Feed are all required to be submitted by May 17, 2013.

Final technical report - due May 19, 2013

This report must summarize the completed rover itself and the whole development process. The report must be between 10-15 pages long, and must detail rover specifications, an overview of its production, the team's rover testing approach, and details of the team's education and public outreach events and activities. A poster presentation will also be required for when the team attends the 2013 Robo-Ops Competition in Houston, Texas.

B. Body Design

Due to the immense amount of research required to size a new robotic platform, our team decided it would be in our best interest to scale an existing and proven platform using its aspect ratio. The platform chosen was XRL, a hexapedal robot currently being utilized by the STRIDe lab.

Our rover's major limiting dimension was the width, due to the size of the motors required to move this platform. The frame was then scaled using a width where two motors could be mounted across from one another and still have an appropriate clearance. From this, the length of the rover as well as the length of our legs was decided. The height of the frame was then minimized to be as small as possible while still allowing us to house all major electrical components in the interior of the frame.

The material chosen for the frame itself was 6065 hot extruded aluminum tubing with dimensions of 2 cm x 2 cm and a wall thickness of 2 mm. This provided the lightweight properties desired while still being able to undergo the stress and impacts of laying the robot down on rocky surfaces. This tubing was welded together, and the aluminum motor mounts for locomotion were welded to it as well. Along the underside of the frame are quarter-inch strips of aluminum that the electrical components will attach to via bolts and straps. This frame will be drilled for all components to attach to the top surface as well as for covers on all sides, as a sealed frame has been decided upon.

The final solid-modeled assembly of the rover's frame weighs less than 5 kg, which is acceptable for the competition weight requirement. The tubing was purchased from our sponsor, Misumi USA, with an in-store credit, allowing us to keep our monetary grants for larger purchases. The frame's final dimensions are 72 cm long x 59.4 cm wide x 10.2 cm tall.

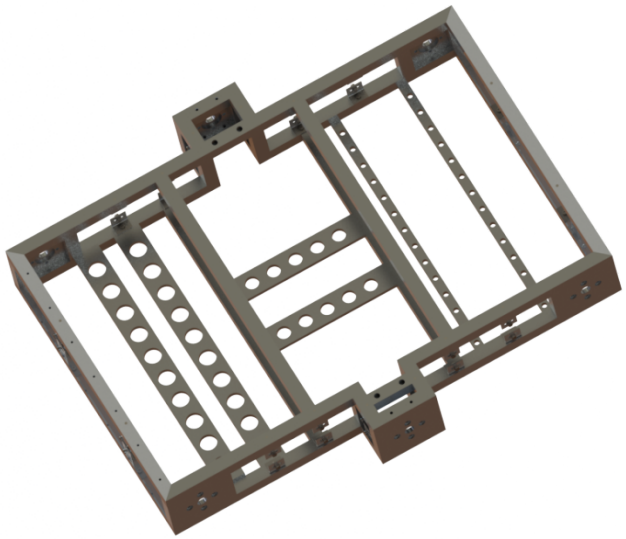


Figure 5 - The frame assembly with all mounting holes and brackets

As mentioned before, the team has decided to have a sealed body. This was accomplished by encasing all sides with eighth-inch ABS plastic, which would keep the weight of this paneling to a minimum. The sides of the rover and underside will have the ABS bolted to them and will be caulked to provide a water-resistant seal. However, it is still necessary to access the internal components of the rover for testing, charging of batteries as well as fixing of any issues encountered. Because of this, the top has been designed to use a series quarter-turn screws pushing down on a gasket to provide the seal between the various panels and the frame. These screws will allow quick and tool-less access to the internals at any point in time. Figure 6 shows the frame with its panels in place with the cut outs for various pieces to be mounted to the top. The addition of these panels and fasteners brings the weight of the now sealed frame up to 7.6 kg.

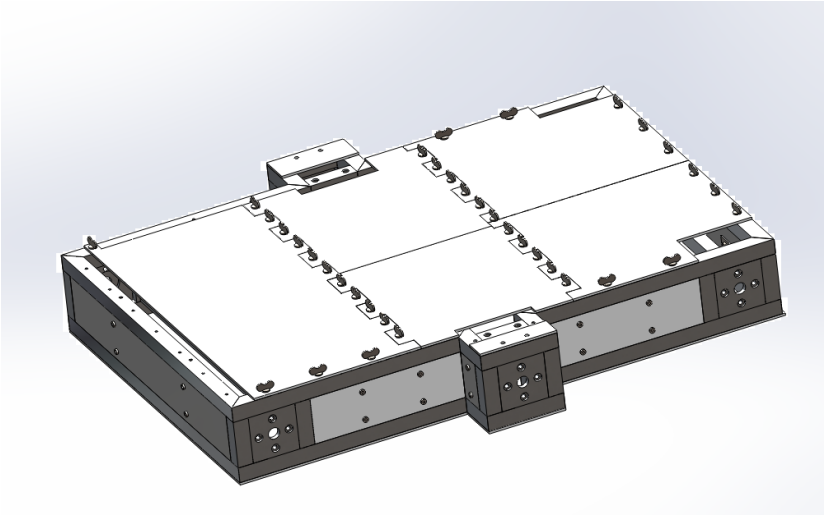


Figure 6 - The assembled frame with all ABS panels and mounting hardware

C. Drive Motors

Selecting the drive motors for our rover was a critical task, which then dictated the size of the rover and the proposed budget for all other systems, as they would be our most expensive purchase. To estimate the motor specifications that we would need, we turned to a research paper being done on dynamic scaling by our sponsor, the STRIDe lab. Using this information, we were able to obtain the desired data, in Table 1, by scaling from the XRL platform. Our sponsor recommended the desired rotational speed as it is a fast walking gate, where the stiffness of the legs of the robot does not become problematic.

	Continuous Torque	Stall Torque	Nominal Rotational Speed
XRL	2.3 Nm	30.52 Nm	187 rpm
Desired	17.1 Nm	226.74 Nm	120 rpm
Selected	17.4 Nm	383.56 Nm	132 rpm

Table 1 - The torque values of the XRL platform, the scaled desired values and the values of the selected motors.

In looking for motors that could produce the desired specifications, there was found only one company that could provide the motors in a relatively small and lightweight package. We ended up selecting Maxon Motors as our provider of this component of our system as they offered an educational discount to our group. To achieve this amount of torque and rotational speed, a RE-50 motor was chosen. This motor is a 24 Volt, 200 Watt motor that is coupled to a planetary gearbox with an encoder mounted to detect the motor's position. This combination provided the specifications in Table 2.

Supply Voltage	24 V
Power	200 W
Weight	1.9Kg
Encoder Counts/Revolution	108,500
Overall Length	207mm
Discounted Price (each)	$867.75

Table 2 - This table outlines the other key specifications of the selected drive motors.

These motors will cost the team a little over $6,000 for seven motors, including a spare. This is our largest purchase for the project but there was no other reasonable alternative. We also can rest assured that there will not be issues with these drives as Maxon is regarded as the industry standard and are known for their reliability.

D. SEM Design Requirements

The competition guidelines give few explicit requirements for the method by which teams pick up the samples (rocks), only specifying that:

"*The rover must be capable of picking up irregularly-shaped rocks with maximum diameter of 2-8 cm and masses ranging from 20-150 grams. The rover must be capable of picking up at least five of these rocks and transporting them using the rover throughout the course.*"

In addition to these explicit requirements, the team has come up with several others to maximize the effectiveness of the Sample Extraction Module, or SEM. The team's primary design criteria will be presented here along with their justifications.

1. **The system must be easy to control.** During the team's concept generation phase, much research was conducted on the effectiveness of sample extraction designs and techniques from the teams of previous years. It was observed that the teams which performed the best in the competition either had low degree of freedom arm designs, such as the winning team from Worchester Polytechnic Institute who used an arm with only two rotational joints, or extremely sophisticated control schemes such as the second-place team from Caltech. At the advice of our advisor, Dr. Clark, we chose to avoid designs with multiple rotational joints and complicated dynamics, as they can take months to refine to acceptable precision.
2. **The reach of the system must be sufficient.** A legged robot cannot be as deliberate in its forward and backward movements as a wheeled robot can, as it is constrained to movement in finite increments. This means that the reach of the sample extraction system must be great enough to account for the minimum step size of the robot, so if the robot has a minimum step size of 8", the SEM must have a reach greater than 8" to ensure the sample can always be picked up.
3. **Autonomous operation of the system must be feasible.** Due to the substantial distance that will be present during the completion (Tallahassee, FL to Houston, TX) a significant time delay will be experienced between the user input, response of the robot, and seeing the action on the screen. As such, any actions that can be automated will save a significant amount of time. The SEM only needs to be user-controlled to navigate the claw to the sample of interest and to pick it up, the entire storage process of the sample can be automated. We looked for designs that would make this automation easy.
4. **The system must be reliable.** The design must allow for highly repeatable motion, low chance of failure, and the strength to survive the bumpy ride atop the legged rover.

Other noted design metrics are manufacturability as the device must be made in house by our machine shop, price to accommodate our budget, and weight to ensure we do not exceed the competition weight limit.

E. SEM Detailed Design

The chosen design for the SEM combines all of the above metrics to result in a design that should accomplish the task of picking up rocks samples both quickly and reliably. The proposed design takes advantage of the fact that the legged locomotion platform results in the body of the robot being non-planar. The vertical degree of freedom generally found in robotic arms was removed from the arm itself and compensated for through the use of the legs of the robot to vertically adjust the end effector. The arm employs a simple and reliable lead screw setup for motion along the width of the robot (X-axis), and a long-stroke linear actuator to move the end effector along the length of the robot (Z-axis). A cam-follower system is used as a passively-activated degree of freedom which allows the claw to rise over the sample bin during storage. Each of these sub-systems will be presented in detail in this section. All of the parts that comprise the system can be seen highlighted in red in Figure 7.

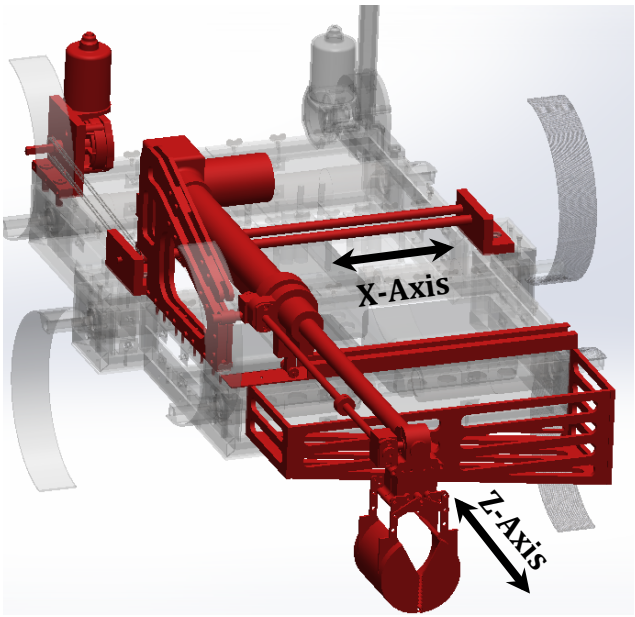


Figure 7 - Sample Extraction Module on rover

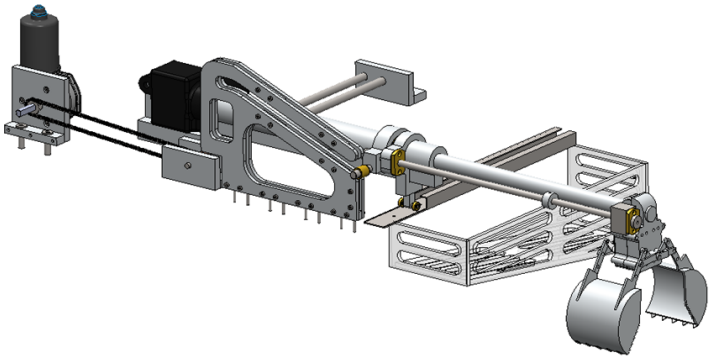


Figure 8 – Isolated Sample Extraction Module

X-Axis Movement

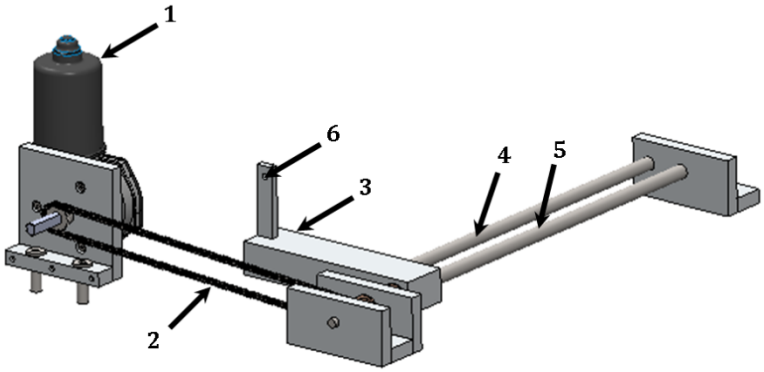


Figure 9 – X-Axis drive system

The x-axis drive system can be seen in Figure 9 and is a basic linear motion drive system based around an ACME ½" lead screw which offers a 1 inch-per-turn speed ratio (4). The lead screw meshes with a lead screw nut inserted into the carriage (3), which results in linear motion of the carriage whenever the lead screw is turned. The lead screw is driven by a high-torque geared DC motor (1) which offers 27 Nm of nominal torque. Connection between the lead screw and motor is made by a chain drive system comprised of a MISUMI chain and sprocket set (2) with a 1:1 speed ratio. The chain drive system was required due to spatial constraints; there was not enough room to couple the motor shaft to the lead screw directly. A 12mm diameter stainless-steel linear support shaft (4) guides an LMU12 linear bushing which is pressed into the carriage to ensure smooth motion. Double-shielded ball bearings are pressed into both mounts to support the lead screw, which will be machined down to 8mm at both ends to accommodate the drive sprocket and optical encoder.

A lead screw was chosen over a ball-screw because it does not require lubrication and has no internal moving parts which can be damaged by dirt or grit. A lead screw also undergoes far less backlash than a ball screw, resulting in more precise and more predictable system operation. All mounting plates will be machined in-house out of ½" aluminum. The mounting plates, carriage, linear support shaft, and bearings were all sized with a factor of safety of 2-6 based on Finite-Element Analysis, which will be described in Appendix I.

Table 3 – Important X-axis performance metrics.

X-Axis Performance Summary	
Movement Speed:	1.5 in/s
Traverse Distance:	12"
Drive Torque:	27 Nm
Positioning Precision:	512 counts/revolution

Z-axis Movement

Movement in the Z-axis, or "lengthwise" axis, is controlled by a long-stroke linear actuator, and mounts to the X-Axis carriage at (6) in Figure 9 which is a pin joint that allows the linear actuator to pivot. The specifications and dimensions of the linear actuator are listed in

Figure 10 and Table 4.

Table 4 – Z-Axis Performance Specifications

Z-Axis Linear Actuator	
Extension/Retraction Speed:	1.5 in/s
Useable Extension:	12"
Fully Extended Length:	34.9"
Fully Retracted Length:	20.9"
Actuating Force:	200 lbf
Rated Voltage:	12V

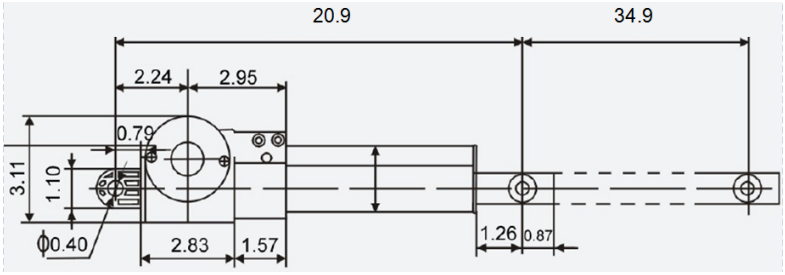


Figure 10 – Actuator dimensions in inches

The linear actuator is DC motor driven and can be controlled using the same pulse-width modulation as the drive motors.

Sample Storage System

The planar arm design requires special considerations with respect to the storage of the sample, as the gripper needs to rise above the storage container in order to drop the sample inside. We investigated a sliding box design which would allow the box to lower enough for the gripper to drop sample inside, but found that the ground clearance of our rover was reduced to below the 10 cm minimum allowed by the competition. Our solution to the problem of raising the gripper is the addition of a passively-activated degree of freedom in the form of a cam-follower. The goal of the cam-follower is to couple the retraction of the linear actuator with pitching about the base of the actuator. The components that comprise the sample storage system are shown below.

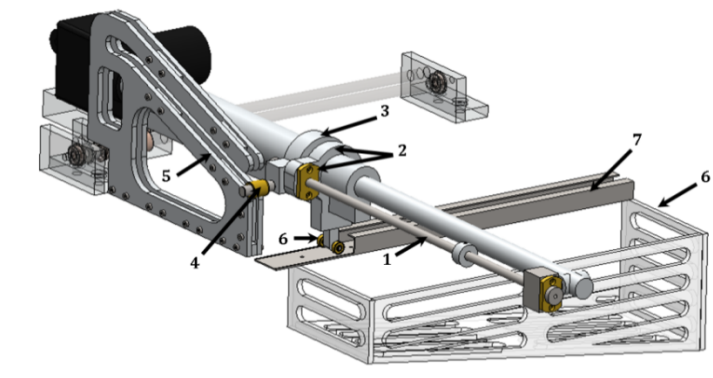


Figure 11 – Sample storage system

Operation of the sample storage system is fairly straightforward and will be automated. Once the sample has been captured in the gripper, the X and Z axes are positioned in the extreme "right-extended" position, as show in Figure 11. Storage is initiated by the retraction of the linear actuator (or Z-Axis), which causes the pushrod (1) to begin sliding through the linear bushing pressed into the fixed bracket (2). The sliding bracket (3) is attached to the end of the pushrod and thus begins to move backward also, this will cause the follower (4) to begin ascending the cam-slot (5). Once the linear actuator has been fully retracted it will be in the full pitch up position, shown in Figure X b), and the gripper will open to drop the sample directly into the storage bin (6). The process is then run in reverse to lower the gripper.

In order to ensure that the linear actuator cannot pitch and the gripper movement remains planar at all times other than storage, a rail (7) was designed for the front of the rover which constrains the rollers (6) attached to the fixed bracket (2). The rail/roller combination will prevent upward movement of the Z-Axis when the rover is not in the extreme "right-extended" position. Stainless-steel was chosen for the rail based on the failure of aluminum during an FEA simulation.

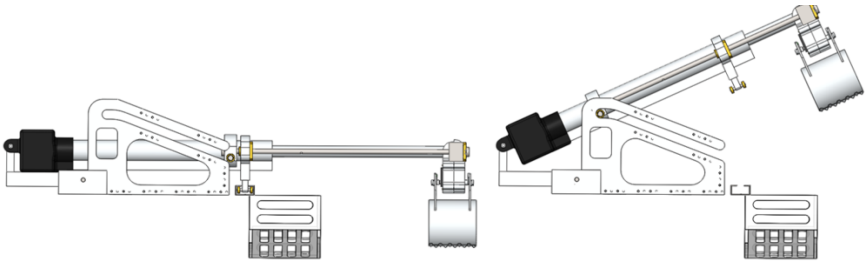


Figure 12 - a) Storage procedure starting position b) End position

F. Gripper Design

Overview

The sample extraction system needs a mechanical component to grasp rock samples at the end effector of the robotic manipulator. According to the competition guidelines, rock samples will vary in sizes and masses ranging from 2 - 8 cm in diameter and 20 - 150 g mass. The mechanism must be capable of acquiring largest rock sample discretely as points will be awarded for the selection of specific rocks. The component must be versatile in rock acquisition, and strong enough to endure competition environment.

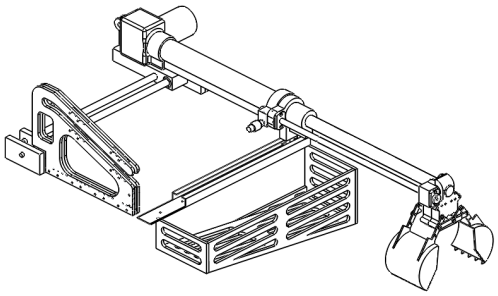


Figure 13: Complete Sample Extraction Module

The mechanism has been termed a gripper and is to be mounted to the end effector of the Sample Extraction Module (SEM, see Figure 13). In order for this gripper to mesh with the sample extraction system and the rover as a whole, it must be capable of interfacing with the main computing hardware. Since this component is also part of a remotely controlled system, it must not require more complexity to control/manipulate than the rest of the system's components. This is to maintain the fast sample acquisition times the team desires. This gripper must not consume too much power while operating, such that it does not jeopardize the rover from meeting the 1 hour operation time requirement of the competition.

Core Performance

The core system of this gripper is based on a single-actuated four bar mechanism. This mechanism will effectively consist of grounded crank and rocker links connected by a coupler link. The coupler's motion is used for actual grasping, and this motion is optimized for large sample acquisition. Each crank is ideally driven by a single servo for positional control. Most commercial hobby grade servos can easily interface with wide variety of computing hardware, and relatively simple control algorithms may be employed to control them.

The grasping geometry consists of a hybrid design capable of both pinch and scoop action. Each pincer is a concave semi-enclosure, similar to an excavator bucket, attached to the coupler link. These buckets meet at their tips to create "pinch" action. When closed, the interior volume can potentially fully enclose a rock sample.

Preliminary Design

A solid model of the gripper design is pictured in Figure 14, below. The four bar mechanisms are planned to comprise of Aluminum links to keep weight down and strength up. The coupler-buckets are connected to a base enclosure via mirrored mechanisms on each side of the enclosure. The Driving Fourbar mechanisms are driven by two servos; one servo per bucket-driving mechanism. The Driven Fourbar mechanisms are passive, and exist for structural purpose only.

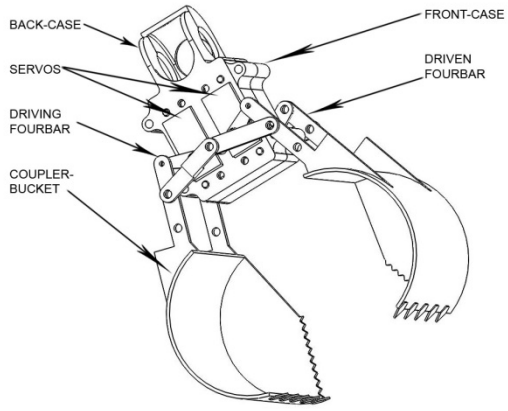


Figure 14: Solid model of gripper with key components annotated.

The base enclosure is a two halved enclosure that connects the gripper mechanism to the linear actuator in the SEM. The whole enclosure mounts directly to the linear actuator and houses the servos that rigidly mount to both halves of the enclosure. Passive cranks and all rocker links mount directly to the enclosure halves, and are to be fastened together via machine screws. The servo-driven cranks mount directly to the servos themselves. Link mounting holes are extruded from the enclosure to account for the spatial property of the mechanism.

Several grasp-enhancement features have been added to help improve the sample acquisition performance. Figure 15, below-left, shows the bucket mechanisms only and highlights two key features. Each bucket features interlocking teeth along the pinch-edge to increase clamping pressure when picking-up rocks. The sides of each bucket are serrated to increase friction if the sample is larger than the gripper, lengthwise. The gripper mouth opens to 14 cm wide, and interior bucket volume is large enough to enclose a 10 cm diameter sphere. Finally, a window will be placed on the rover-side faces of each bucket to enhance viewing of sample capture.

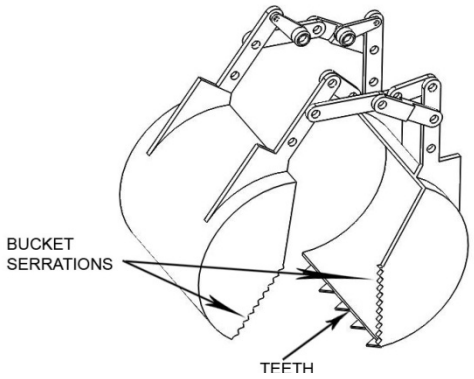


Figure 15: Bucket-driving mechanisms showing a couple enhancement features. The back void is to be replaced with clear viewing windows.

Final Design

Though the final gripper design is still based on the hybrid pincer/scoop design it utilizes a chain drive and gear arrangement to achieve the desired motion rather than a rigid linkage. Figure 16 shows the finalized gripper design.

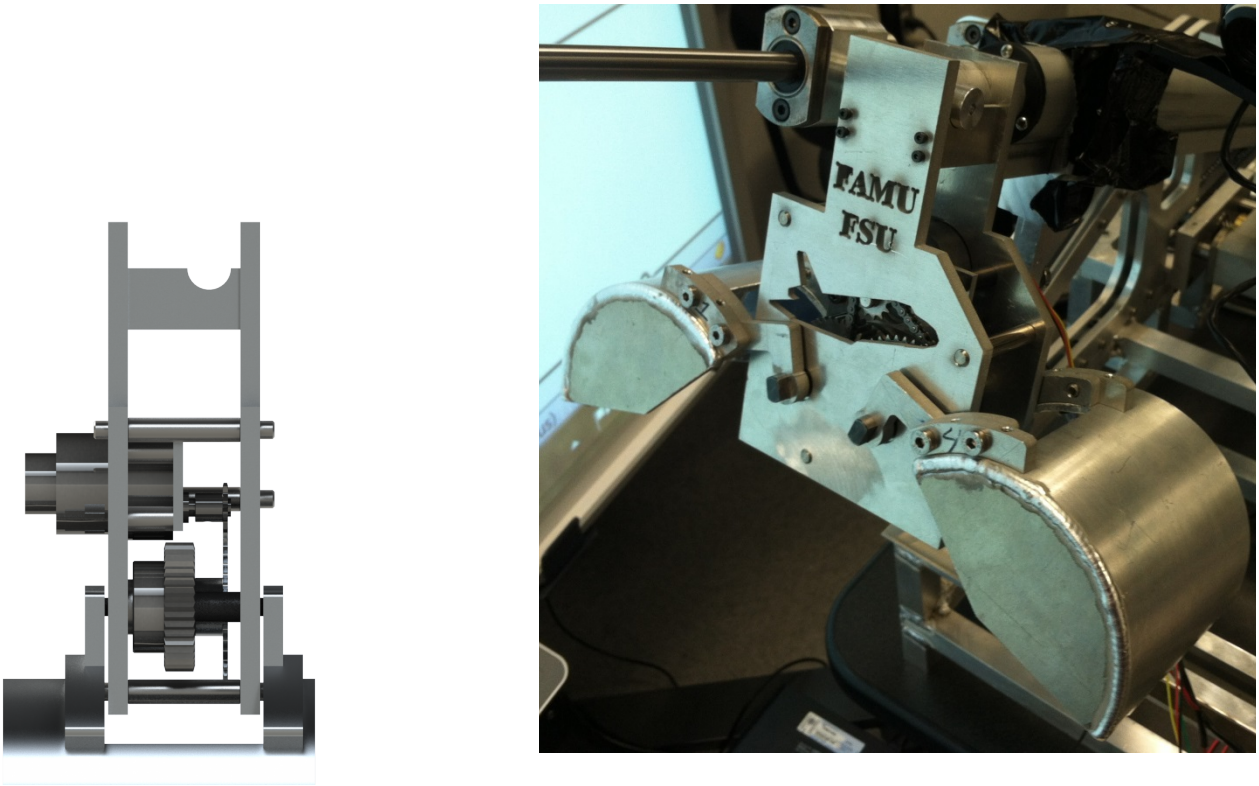


Figure 16 - CAD view of finalized gripper profile view depicting chain drive and gear train (left). Photo of the gripper in the open position (right).

The design is powered by a Lynxmotion 7.2V gear motor coupled to a 4:1 gear train to increase the output torque to 1.4 Nm. The output sprocket is directly coupled to one of the bucket drive shafts. A 1:1 spur gear arrangement couples the two drive shafts together to achieve the desired counter-rotating motion. The arms which connect the drive shafts to the buckets are dimensioned so that the buckets sit directly above level ground in the clamped position. The original arms resulted in clamping just below level ground; testing revealed that the motor could not penetrate even moderately hard soil so the arms were shortened. The maximum clamping force of the gripper is approximately 5 lbf at the bucket tips, which has proven to be sufficient to grip rocks firmly and to penetrate soft soils and dry sand.

G. Sample Extraction Module Control

Electronic Hardware

At the heart of the sample extraction module's control is the Raspberry Pi computer running Arch Linux. This computer holds all of the control algorithms and logic to use the SEM, which is all coded in C. The Raspberry Pi communicates with two motor controllers via UART RS-232 TTY style serial communication. In order to communicate over this communication protocol, the getty process that defaults on this serial line has to be moved to another line. By disabling this process on TTYAMA0, it defaults to TTY1 which frees up the general purpose input output pins necessary for the communication.

The Basic Micro RoboClaw motor controllers connect to the motors of the SEM. The X and Z axis connect to one motor controller (address: 128). On the other of the motor controllers (address: 129), is the gripper motor and encoder. By specifying the address as the leading byte of the serial command, the correct motor controller will respond and actuate the appropriate motor.

Software

The functions written for the SEM are described below. The first functions are the relative move in either the X or Z axes. These allow for the user to call the function and specify a distance in inches to move in either direction. Built into these functions are safeties so that the arm will not try to extend past its limits. If a command is given to move past a particular limit, the actuator will move until its limit and stop, letting the user know that it only moved until its limit and provide its exact position.

The initial function that the user will call when planning to acquire a sample is the deploy function. This function is an automated process that accepts desired end coordinates in inches. This function then takes the arm from its stored position at the top of the cam-follower slot to its full extended (Z-axis) position and then moves to the user specified coordinates after checking that the moves are possible. This initial movement allows the arm to fully ride out of the follower and then engage in the vertical movement limiting tubing across the front of the rover.

Once the arm has been moved above the sample, the user has three options to control the gripper. By calling the gripper function and specifying a '2' after, the gripper closes until its claws touch and then resets the decoder register to provide a reference for open and closed, this should be done at the beginning of the competition. For normal use, the opening function moves each of the arms 90 degrees from closed and then stops and holds them open. In order to grip or enclose a rock in the gripper, a separate algorithm was written to allow for either scenario. When the number '1' is specified after calling the grip function, the gripper closes until it has stopped moving (either two clamshell buckets touching one another or pinching a rock) and then stops the motor.

Once the sample has been extracted, there is an automated return home process that can be called which returns the arm to the fully extended Z-axis position and zero X-axis position. From there it returns to its origin at the top of the follower and can safely deposit its sample into the front mounted bin.

H. Camera Boom

During early vision tests, it was concluded that out main camera used to navigation and finding the target samples needs to be elevated above the rover. A low point of view restricted the ability to depict the colored samples in various terrains. In order to achieve the elevated view point, the rover's main camera will be located at the top of a camera boom. Due to the stored size constraints of the competition, the camera boom cannot be permanently fixed in its vertical position. The proposed design uses a single geared motor to raise the boom from the stored position to the vertical position where it will remain throughout the entire competition. When the boom is in its stored position it is laid horizontally across the top of the rover. At the start of the competition, the boom motor will be activated. When the boom has reached its vertical position it will come into contact with a backstop attached to the motor mount. The back torque on the motor will cause a sudden increase in the current supplied to the motor. The operator will be able to view current and voltage supplied to the motor which will be transmitted back from the rover. When the operator detects the increase in current, the power to the motor will be cut off. An alternative to the operator manually cutting off power to the motor is a programmed command which performs the same action, or a limiting switch which is activated when the boom reaches the vertical position. The optimal power cut off procedure will be determined during testing. The implementation of any power cut off method discussed negates the need to visually see the position of the boom, and the need of a position encoder which simplifies the rovers overall control algorithms.

A motor which was used for a different purpose by last year's Hexcavator team has been selected to raise the boom. The motor can produce a torque output of 23 N·m which is more than sufficient than the required 6 N·m required to raise the proposed boom design which is 62 cm in length, the camera mount, and selected camera. The main benefit of the selected motor is the gearbox it uses. The gearbox consists of a worm gear on the motor shaft which is mated to a spur gear on the gearbox output shaft. This gearbox design is extremely hard to back drive, meaning induced moments about the output shaft due to walking motion of the rover and the length of the boom will not be great enough to cause the boom to lower. A back stop is used to further increase the stability of the boom. The design does not have an active camera stabilizing system due to the large size and weight of systems currently available.. Figure 17 shows the camera boom design mounted on the rear left corner of the rover

Figure 17 - Camera boom mounted on the rear left corner of the rover.

I. Finite Element Analysis

The finite element method provided us with the verification of the strength of materials selected while ensuring us that our rover design would remain under its weight restriction. This analysis was done on several key components, which have been detailed below.

Body

The body of the frame was the first component that this method was applied to. While previous year's efforts to build a walking robot used aluminum tubing that had a thickness of over 3mm, we believed that weight could be saved in the making of a new frame. In the rovers walking process, the frame undergoes very little loading so we decided to verify the strength by putting the frame in far more extreme situations that would be possible. In Figure 18 one can see that the frame still has a factor of safety of 4 while undergoing a load of 50 lbf being pushed up on one of the robots legs. While this may seem as though a thinner tubing could have been used, the college of engineering machine shop has stated that they cannot weld any tubing

thinner than this. It is also useful to have a large factor of safety on this part as it will take any impact from the terrain as the rover lays itself down.

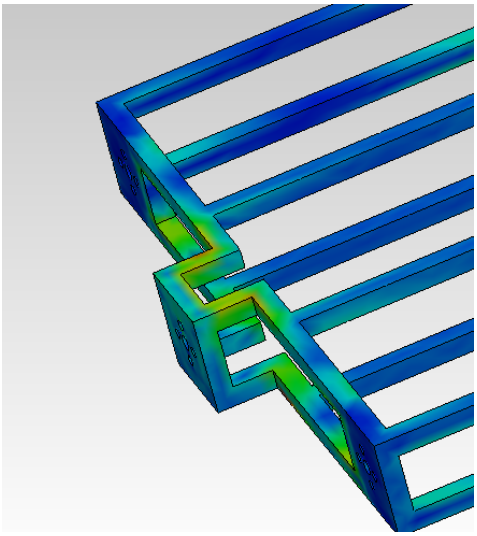


Figure 18 - The testing of the maximum conceivable force on a leg still provided a factor of safety of 4.

SEM Components

The sample extraction module has several components that were significantly influenced by the testing of these parts in simulation. The linear support rod, cam-follower rail and front guide rail all had finite element analysis done to assure the proper strength and weight properties.

The linear support rod is a critical part of the sample extraction module as it helps support the weight of the linear actuator and its load. It also removes the rotational degree of freedom the linear actuator's carriage would be capable of with just a lead screw in place. In doing this testing, the team was looking for a clear material choice, diameter and whether a solid or hollow rod would be best suited. In doing this testing, the conclusion was reached to use a solid steel rod 12 mm in diameter. This allowed for the proper amount of support from the rod. The hollow rod was decided against for its added dimensions to the carriage, as it would have to have been bigger. The simulation can be seen in Figure 19 below.

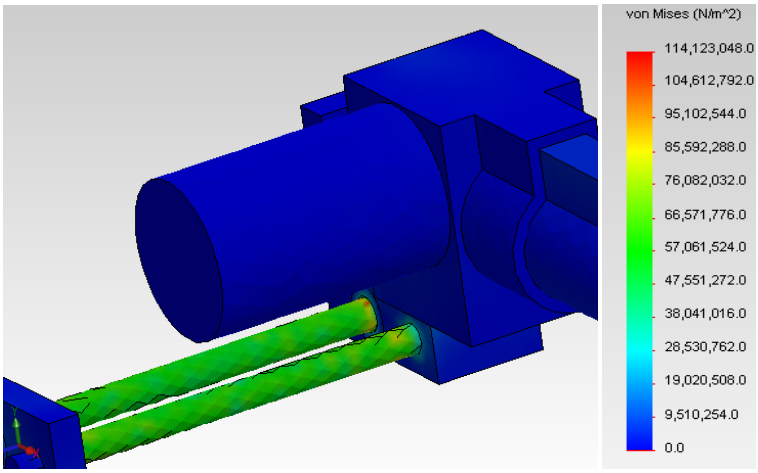


Figure 19 - The linear support rod is able to hold the weight of the actuator with a factor of safety of over 4.

The cam-follower rail was the next item of the sample extraction module to be tested. The purpose of this test was to ensure that the maximum load of the linear actuator (200 lbf) could be applied to the rail without failure while keeping weight to a minimum. Through the repeated testing of this part, the design of the holes machined into the rails was refined until a factor of safety of two was reached. The weight of the assembly was ultimately reduced to under 1 kg. The maximum stress occurred at the front most mount to the frame as seen in Figure 20. If for any reason, this factor of safety needed to be greater, the mounts to the frame could be made out of steel, as opposed to the aluminum mounts currently in use.

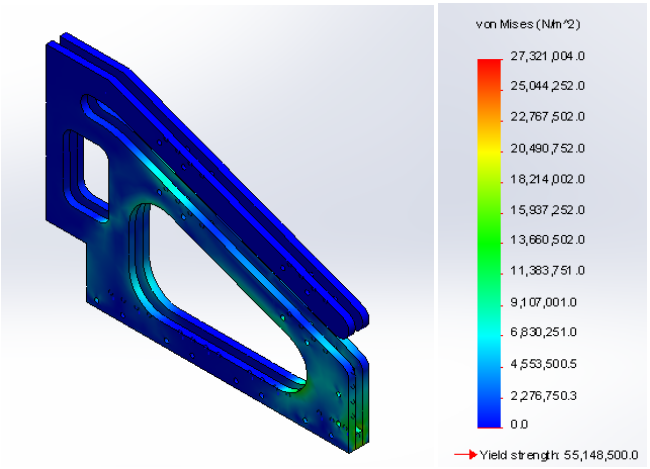


Figure 20 - The cam-follower rail loaded to 200 lbf still provided a factor of safety of 2.

The last component of the sample extraction module that undergoes large amounts of stress is the front guide rail. Due to potential video lag in the competition, the user may lay the rover down on the gripper. In this case, the rover would have half of its weight (max: 50 lbs) placed at one point along this rail. In testing, it was determined that the aluminum rail proposed would not be able to handle the stresses necessary. The

team then decided to use a stainless steel rail to handle this. As seen in Figure 21, the rail has a factor of safety of over 6 in this situation.

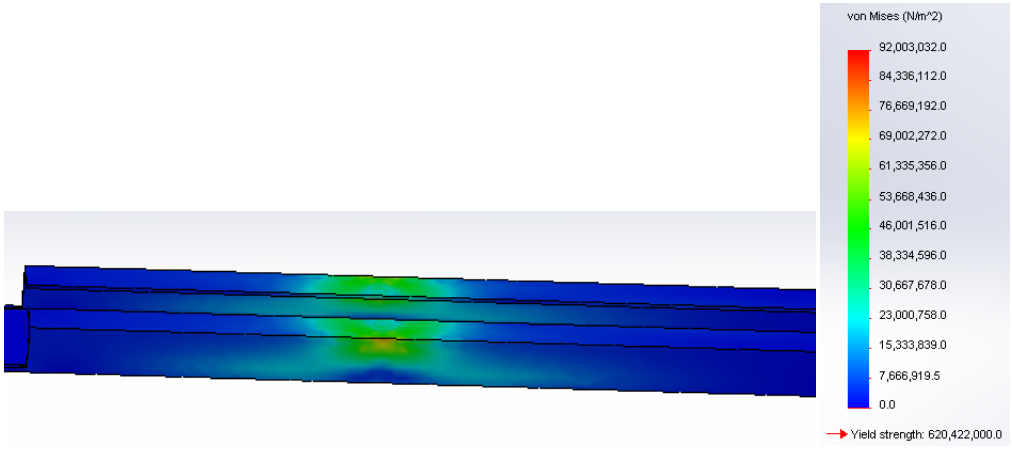


Figure 21 - The front guide rail with half of the rover's weight at a single point still had a factor of safety of over 6.

J. Prototyping

In order to get a proper feel for our design, the team decided to prototype early in the fall semester so that problems could be discovered at an early stage. This later allowed the team to start the machining process earlier and with fewer problems. The prototyping consisted of building a mock frame and validating the design of the planar arm concept.

The frame dimensions were originally decided upon in the middle of September. Once this decision had been made, the team built a to-scale version of this frame out of wood to get a better feel for the size and how components could be housed within this frame. In this prototype were also place holders for size of the motors proposed at that point in the project. After some careful consideration, the frame was scaled up further to its current stage to help house the now larger motors and additional battery sources.

The validation of the arm design was done using a linear actuator and dc motor left from last year's team. They were attached to the mock-up frame created so that the proposed motion could be tested. This confirmed that the control of such an arm would have quick and easy control as originally thought. Due to this testing, we were able to move forward with the design and shortly after order the parts for this system. Both the frame and the arm can be seen in Figure 22.

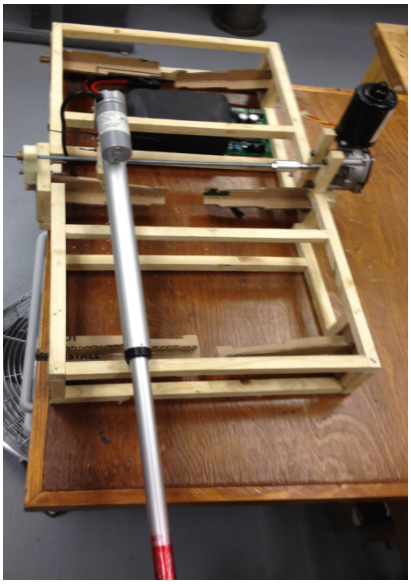


Figure 22 - The prototype of the frame and arm concept helped with design validation and visualizing placement of components.

The electrical and control side of this group also had to have a platform to test their proposed equipment on. This was done using last year's platform known as Hexcavator. Here they were able to test their new control hardware and algorithms without having to wait for the proposed rover to be built. The main advantage of using this platform to test was that it used similar motors to those being used on the proposed rover and the exact same encoders. This would ensure that there will be a relatively seamless transition from

one platform to the next. This platform was stripped of its old control hardware and is pictured in Figure 23 with select new hardware installed for testing.

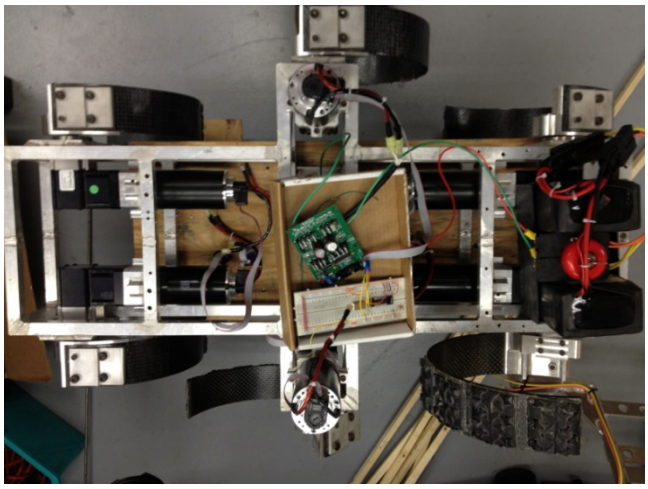


Figure 23 - Last year's platform set up for testing of new hardware configuration.

K. Serial Peripheral Interface (SPI) Protocol

The team stated that communication between the Raspberry Pi and the Xula2 board will be accomplished with the use of the SPI communication protocol. It is a a 4 wire serial protocol that sends one bit of data per clock cycle over the communication lines. It allows for communication between two or more chips. The four wire protocol follows a single master and multiple slave system where the master is defined simply by its absolute control over communication between itself and each slave. The minimum four lines are the master input/slave output line (MISO), master output/slave input (MOSI), the serial clock (SCK), and the slave select (SSEL). The figure depicts a SPI communication system where there are three slaves. Notice that the SCK, MOSI, and MISO lines are all connected to each slave. Also notice that there a multiple slave select lines, each select line allows the bus master to communicate only with the desired slave. Each slave is required to have its own individual select line.

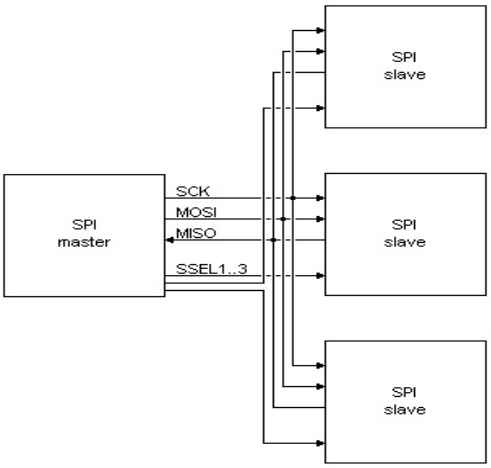


Figure 24 - One master, three slave system obtained from www.fpga4fun.com/SPI1.html

The figure below details how the SPI protocol operates during a single 8-bit data transfer. At 1, the master pulls the slave select line to logic level '0'. This initiates the transfer between the master and that particular slave. At 2, the master begins to toggle the serial clock, once for each bit transferred. SPI is a full duplex system, so for each bit that the master transmits to the slave, the slave transmits a bit of data to the master. At 3, the data transmission is complete so the master pulls the slave select line to logic level '1'. During this entire operation, every other slave select line is kept at logic level '1' so that they ignore the incoming data and transmit no data of their own.

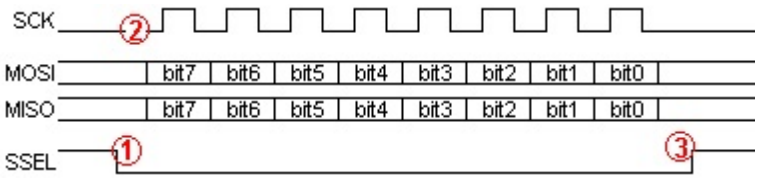


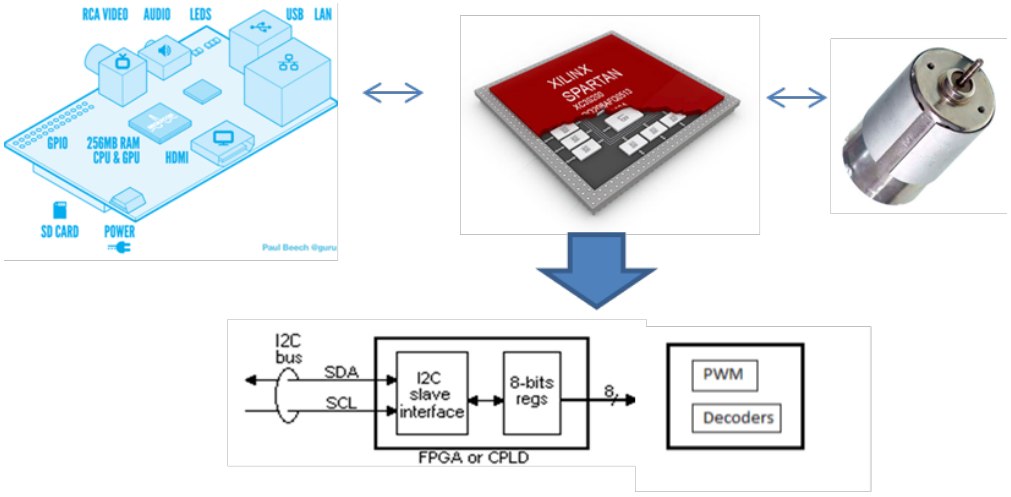
Figure 25 - Typical SPI data transfer operation - obtained from www.fpga4fun.com/SPI1.html

L. On-Board Computing Systems

Raspberry Pi

The Raspberry Pi is a Linux based computer that is about the size of a credit card and is a possible solution for onboard computing for the camera. It has a 700 MHz clock, 256 MB of random access memory (RAM), 2 USB 2.0 ports, an Ethernet port, and power is supplied via 5 V micro-USB port. There are already many peripherals that have been verified to be compatible with the Raspberry Pi; one of the most important being USB 3G dongles. The dongle would allow the Raspberry Pi to connect to a 3G network. Unfortunately the USB ports that are standard do not supply enough power for 3G dongles so a powered USB hub will be required. Since the operating system is Linux, communication with the Raspberry Pi is fairly straightforward with the ability to access its shell account. As long as the IP address is known, the Raspberry Pi can be communicated with and controlled.

However, the Raspberry Pi does not have enough pins to control all 6 motors in both directions and read every decoder. There are several pulse width modulation ICs and decoders available in the market but many of them require too many pins and those that are serial interfaced are not capable of handling 6 motors. Reading several decoders through a serial interface will introduce delays in our control algorithm. Therefore, the appropriate solution is to create the logic using an FPGA. This allows us to create the logic to control and read position from every motor in one compact design. The FPGA will have a serial interface which we will connect to the Raspberry Pi.



M. Raspberry Pi Pinout

The Raspberry Pi has 26 general purpose input/output (GPIO) pins available. The figure below shows the layout and GPIO number of the pins. For the purposes of the team's particular application, GPIO8, GPIO9, GPIO10, GPIO 11, the 5V power and the numerous ground pins are used. GPIO pins seven through eleven are dedicated serial peripheral interface (SPI) bus pins that the team will use to facilitate communication between the Raspberry Pi and the Spartan-6 field programmable gate array (FPGA). GPIO 8 is the slave select

line that the Raspberry Pi will pull low, i.e. send a '0' to that line, to choose a slave to communicate with. GPIO8 will be used to select which of the decoders to receive information from. The rate at which the SPI bus is clocked at is determined by the Raspberry Pi via the SCKL (serial clock) pin. Prototyping with the Altera DE2 board has revealed that the SCKL will be able to achieve a frequency of 32 MHz, which will be more that fast enough for the team's purposes. The Raspberry Pi will receive information of motor positions from the MISO pin.

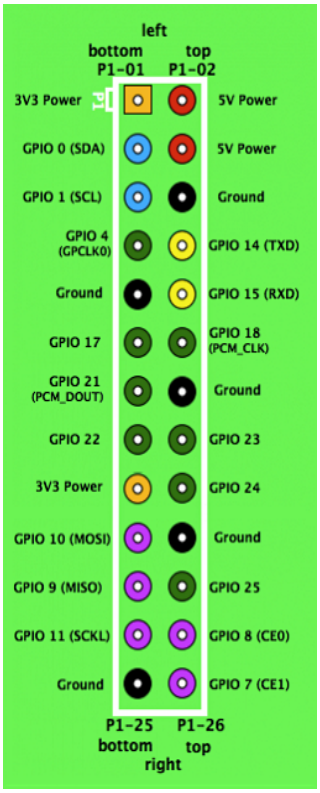


Figure 26 - Raspberry Pi pinout diagram, obtained from www.elinux.org/RPi_Low-Level_peripherals

N. Field Programmable Gate Array Board

The Xess Xula2 board houses the Xilinx Spartan VI - XC6SLX25 FPGA along with 8 Mbits of Flash memory, an onboard 12 MHz oscillator, and a 40-pin interface, 33 of which are general input/output pins. The Spartan-6 has 24,051 logic cells, four digital clock managers, and two phase lock loops (PLL).

While prototyping the FPGA circuit that consisted of six PWM modules, one SPI module, one decoder module, twelve AND gates and six NOT gates, the logic cells that were used was 245. Given that the circuit will be complete with only six more decoder modules, the team determined that 24,051 logic cells would far exceed the necessary logic cells for the circuit. The PLLs on the Spartan VI multiply the frequency of the 12 MHz oscillator to 255 MHz to achieve a 1 MHz PWM signal for precise control of the leg motors. The following equation was used to achieve this conclusion:

$$\frac{f_{clk}}{PWM\ Resolution} = PWM\ Frequency$$

With a duty cycle of eight bits, the resolution becomes 255. Substituting this value and the desired PWM frequency, the equation becomes:

$$\frac{f_{clk}}{255} = 1MHz$$

$$fclk = 255 * 1MHz = 255\ MHz$$

The Raspberry Pi will require five pins to communicate with the FPGA board via SPI; each of the six PWM modules will require two pins to transmit the PWM signal from the FPGA to the motors; and an additional seven pins will be required for the decoders in the FPGA to receive motor position data from the motor encoders. This sums to 24 pins out of the 33 GPIO pins available that the design will require. That leaves nine currently unused GPIO pins for any possible additions to the design before the available pins are exhausted.

O. Sabertooth Motor Driver

Figure 27: Sabertooh Motor Driver

After numerous problems with the OSMC motor drivers, the team decided that it would be in our best interest to procure a new set of motor drivers. The choice was the Sabertooth 2x60 motor driver. As it name suggests, it is a 2 channel motor driver that can provide up to 60 A continuous current in each channel. In other words, each of these motor drivers can drive two motors at the same time at 60 A maximum. The input voltage can be 6-30 V nominal with 33.6 V being its absolute maximum voltage. The driver has four different modes of operation, analog input, R/C input, and two serial inputs that differ only in their complexity. Modes of operation are chosen with the dip switches provided by and attached to the motor driver. There are ten possible connections that the Sabertooth features. Four of these connect to the motors, two to the power

supply, 0V connection and a 5V connection. The last two connections are for general signal input and are called S1 and S2.

Pulse Width Modulation

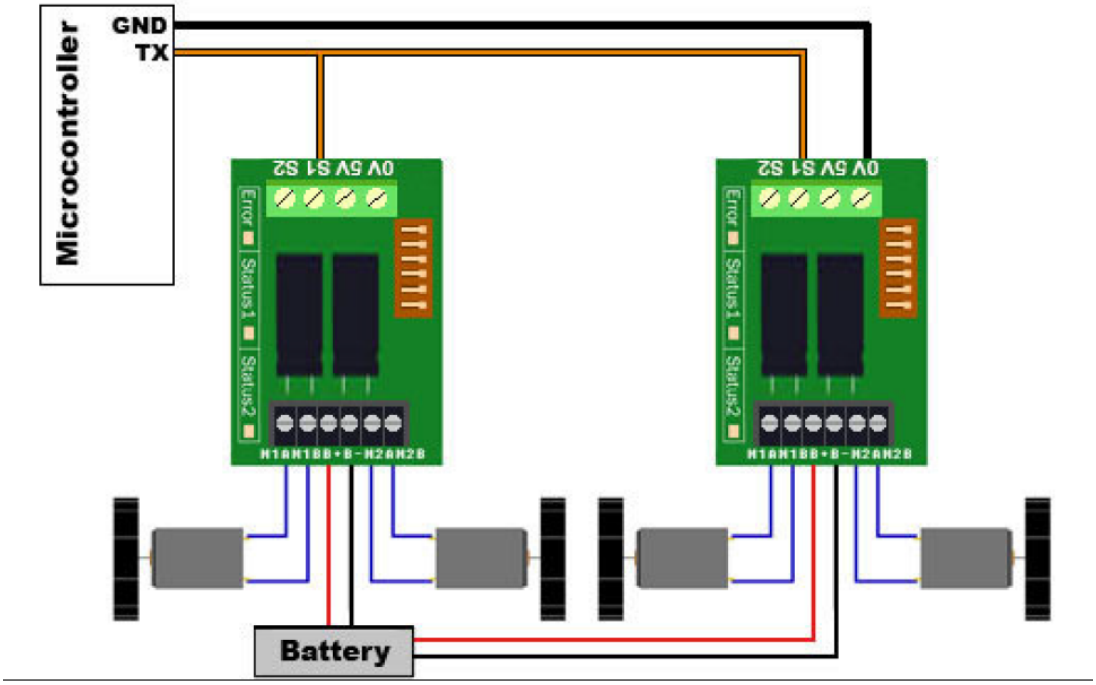


Figure 28: Connection Schematic for Serial Communication

With the introduction of the new Sabertooth motor drivers, it was found that the team's Pulse Width Modulation (PWM) module was not compatible with them. In other words, the PWM module that was designed and implemented with the older OSMC motor drivers was completely useless. Fortunately, the Sabertooth has other means of communication to drive the motors. The motor drivers are communicated with via RS-232 through the S1 connection with packets of information. The nearly four byte packets consist of an address byte, a command byte, a data byte and a 7 bit checksum. The address byte is used for the motor drivers to determine which of the motor drivers are being communicated with. The motor drivers are provided with dip switches for users to set the address of a particular motor driver. The second byte is the command that the motor driver will execute. Since the address byte only choses which motor driver to communicate with the command byte is used to determine which of the two motors will run. The command byte also whether the chosen motor will move forward or backward. The data packet is the desired duty cycle at which to run the motors. The final seven bits serves as a checksum that is used for error detection. If the checksum is not the correct value, the command will not be acted upon. The checksum is calculated as follows

$$checksum = address + command + data$$

In the packetized serial mode, the driver configures S2 as an emergency stop input. This active low

signal is also used by the team in the unlikely case of the rover having to stop suddenly.

P. Field Programmable Gate Array Circuit

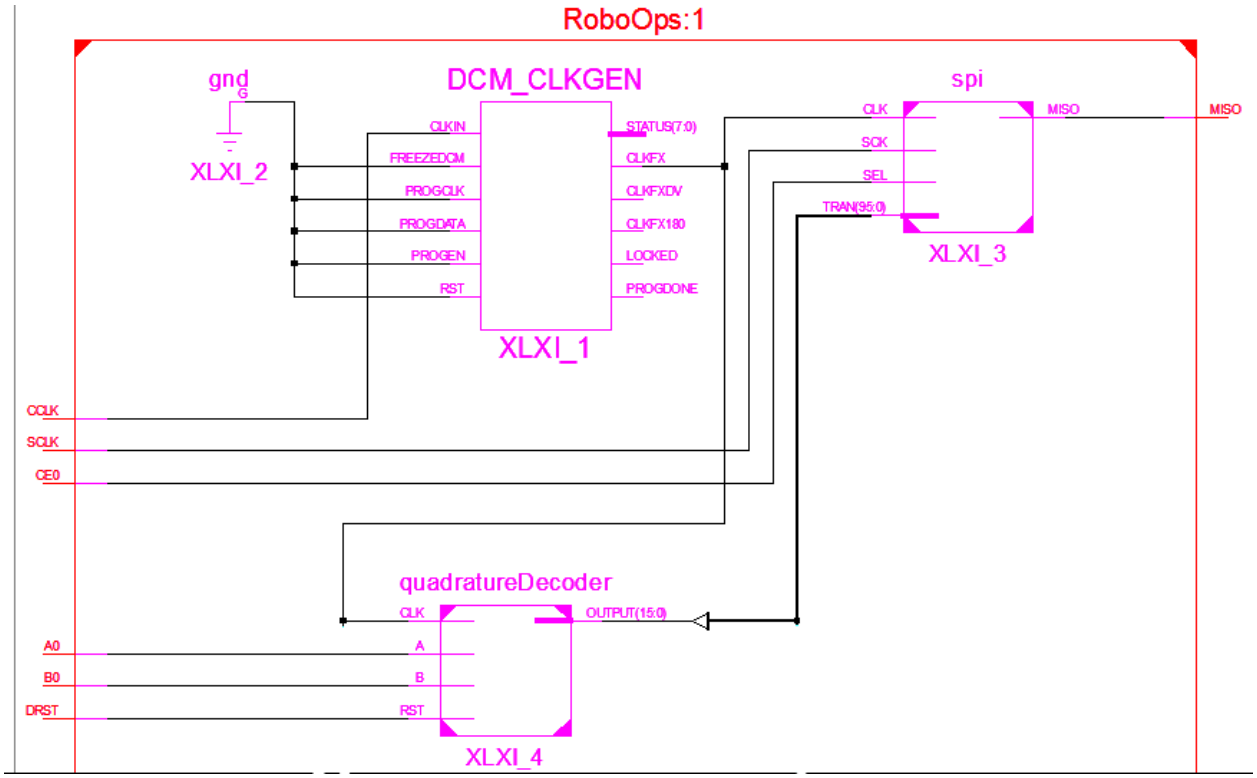


Figure 29 - Top level of the FPGA cicuit instantiated by the Xilinx ISE Register Transfer Level (RTL) Viewer

The figure above gives a top level diagram of a portion of the circuit described in the very high speed integrated circuit hardware description language (VHDL) that the Spartan-6 will be programmed to implement. For simplicity and space, this portion of the circuit only depicts a single decoder module. When communicating with the FPGA circuit, the Raspberry Pi acts only as a receiver of the decoder modules' outputs.

Q. Serial Peripheral Interface Module

The SPI module first synchronizes disparate clock frequencies input from the SPI serial clock and the PLL output clock by oversampling the SPI serial clock with the much faster PLL clock. Without synchronized clocks, the faster PLL clock would read a single transmitted bit multiple times during a much slower SPI clock cycle. The oversampling is accomplished by storing SCK, MOSI, and SEL into three separate two bit shift registers. When SCK changes from '0' to '1', then a rising edge on the serial clock has occurred and data on the MOSI and SEL lines are valid. The module counts the individual bits by counting the falling edges of SCK, that is when SCK changes from '1' to '0', while SEL is '0'. As long as the count of bits received from the Raspberry Pi is less than or equal to 16, the current bit on MOSI will be stored in a 16-bit shift register on the rising edge of SCK. This process works similarly for the current bit being transmitted via MISO except storages into the

shift register occurs on the falling edge of SCK. Since the SPI protocol is full duplex, there must be data being transmitted in both the MISO and MOSI lines. Because the PWM module has nothing to transmit to the Raspberry Pi, the SPI module simply transmits two bytes of zeros to the Raspberry Pi, which will ignore those two bytes. When communication with the decoders is active, the SPI module will transmit two bytes of data representing the position of the motor to the Raspberry Pi.

Slave Selection

Traditionally, the SPI protocol has a single slave select line for each slave in the system. The Raspberry Pi has only two dedicated SPI slave select lines, which is not enough to select six decoders. Rather than using and debugging the "demux" module proposed and prototyped for the Final Design report at the end of last semester, the team decided that a simpler approach would be the more desirable option. Instead of selecting one of the decoders to read from, every decoder is read from at the same time. So when the slave select line goes low, the Raspberry Pi will read all 96 bits (six 16-bit decoders) from the FPGA. The Raspberry Pi is able to discern which reading comes from which decoder by assuming that decoder one's reading is represented by the high bits and decoder six's reading is represented by the low bits. This can be accomplished because the SPI communication with the decoders is set at 8 MHz while the serial communication with the motor drivers is in the kHz range.Decoder

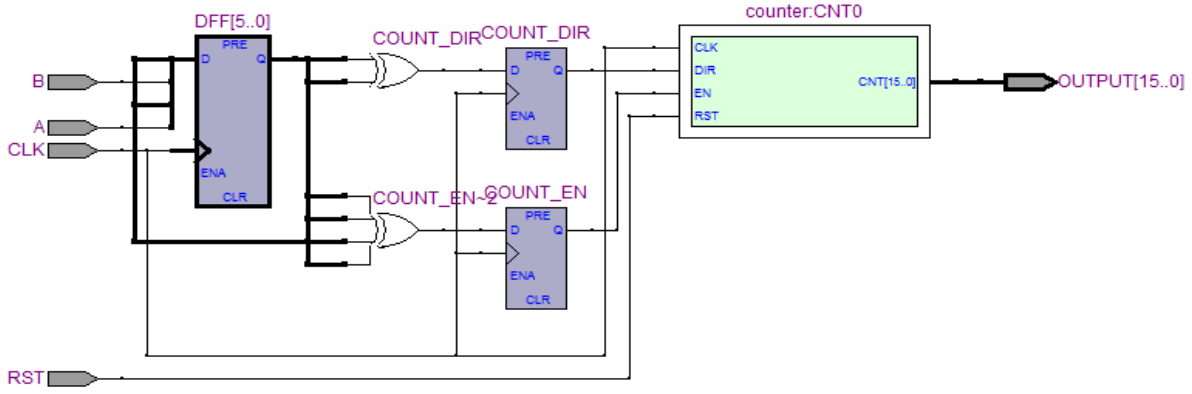


Figure 30 - Decoder logic circuit.

The preceding diagram depicts the decoder hardware that is used to count the quadrature encoded signals provided by the motor encoders. The counter is connected to the counter direction signal, COUNT_DIR, and the counter enable signal, COUNT_EN. If COUNT_EN is asserted, the counter will increment the count by one when COUNT_DIR is logic level '1' and decrement the count when COUNT_DIR is logic level '0'. Input signals A and B are 90° out of phase the count direction is dependent on how the signals are oriented to each other. If A precedes B, then the counter will ascend, if B precedes A, the counter will descend. That is assuming that the COUNT_EN signal is logic level '1'; if it is '0' then the count will not change. COUNT_EN is pulled to logic level '1' when a state change occurs in either of the input signals. The state change is detected by comparing the value of both signals before and after being stored in a flip flop. If the values are different, a change of state has occurred and the counter will become enabled.

The number of ticks in a single revolution of the motor in the prototyped decoder is currently set as 26,000. This requires at least 15 bits to represent in a binary counter and for reasons that will be made clear in the descending case; the counter is set to be 16 bits. This creates slight complications when the 25,999 count is reached since the counter does not automatically overflow and reset to zero. This is solved by comparing the counter value to 25,999. In the ascending direction, if the counter is greater than 25,999, it is set to 0; this would be the 26,000th tick and the motor would have made a complete revolution. In the descending direction, the counter is compared to zero. If it is less than zero, the counter is set to 25,999 and will continue to count down. This is the reason that a 16-bit counter has been implemented. Quartus uses the two's complement format to determine the sign of a number so if only 15 bits were used 25,999 would be considered a negative number since a one would be the most significant bit. In 16-bit binary, though, a zero is the most significant bit and 25,999 would be considered a positive number and the counter would count down.

Decoder Simulation

This is the simulation result from the decoders when provided a quadrature encoded signal on the inputs a_test and b_test. The signal ooutp is the register where the count value is stored. As can be seen in the simulation figure, the output signal, ooutp increments every time signal a_test or b_test changes its state. If the changes in b_test preceded those of a_test, ooutp would decrement.

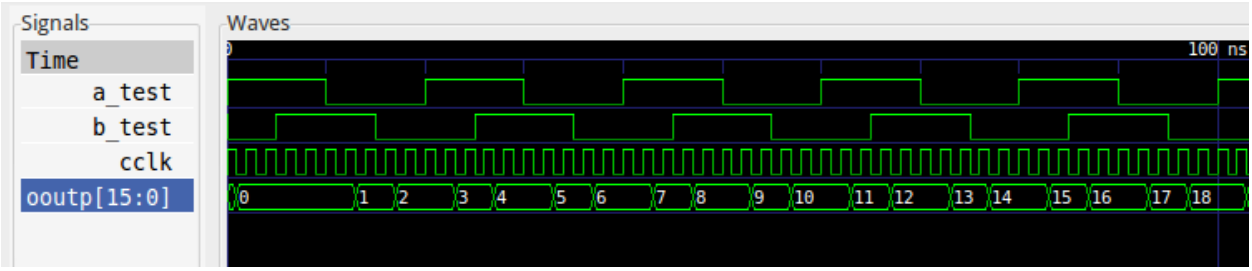


Figure 31 - Simulation waveform of the decoder module

SPI Protocol Slave Simulation

The following diagram is the result of simulation the SPI module of the team's FPGA circuit diagram. For this simulation, cclk represents the Xula2's onboard clock, mmiso is the MISO line, mmosi is the MOSI line, rrecv is the received data register, ssck is the SCK line, sssel is the SSEL line, and ttran is the transmitted data register. The data sent over the mmosi line will be stored by the rrecv register and the data sent over mmiso line will be stored in the ttran register. As can be seen, ssck pulses only while sssel is pulled to logic level '0', exactly what was expected of the SPI module. The master transmits on the mmosi line, from left to right, "1010 1010". In hexadecimal format this is AA, which is what the rrecv line displays after the transmission is completed. The slave transmits to the master via the mmiso line, from left to right, 1100 1010. In hexadecimal

format this is CA, which is what the ttran line displays. Thus based on this and other simulations, the SPI module is determined to be functional.

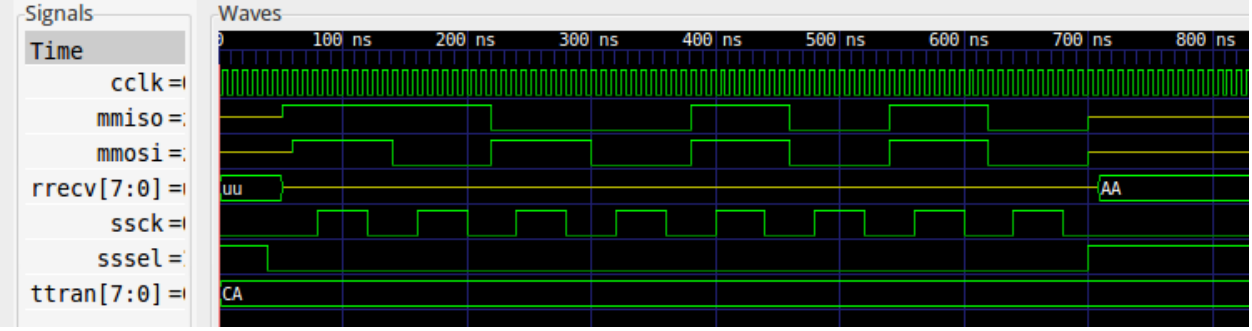


Figure 32 - Simulation waveform of the SPI module

R. Software Development

Development of the Buehler Motion

The Buehler motion was written in to code by using the following Figure.

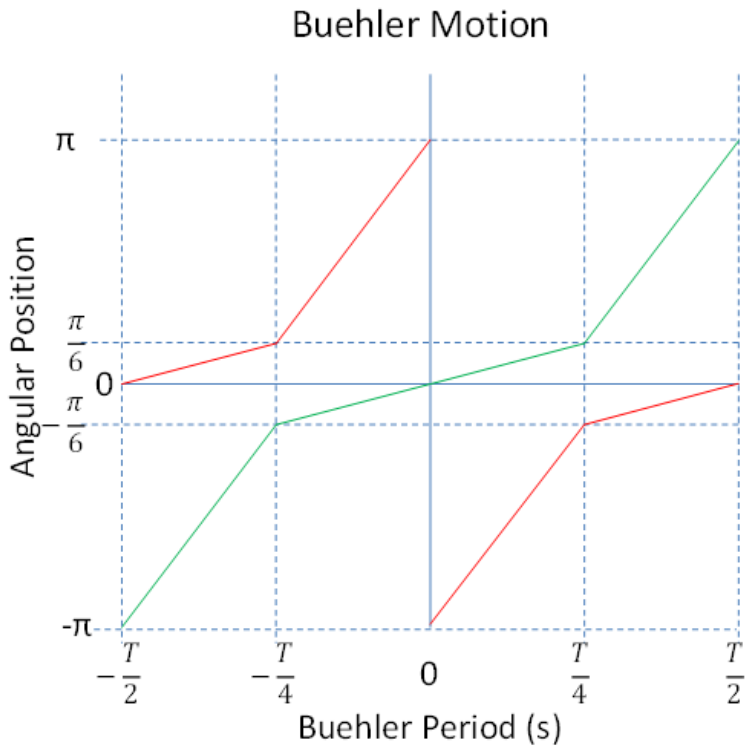


Figure 33 - Mathematical basis for Buehler algorithm.

The goal was to have at least 60° be the slow or step region while the rest of the motion would be in the fast or sweep region. This is depicted in the graph in the strip in the vertical middle of the graph where the green and red lines are in the step region between –π/6 and π/6 angular positions. The independent axis corresponds to the entire Buehler period that is divided in to four sections. The boundaries are kept generic

so that the speed of the rover can be varied. The angular position also is easily transformed into motor position tick by noting that π is the half way mark on any angular sweep. Thus π corresponds to the maximum number of ticks divided by two. Using this information and basic linear line derivation techniques, the following equations were derived:

- For $\frac{T}{4} \leq |x| \leq \frac{T}{2}$
  - G: $\frac{10\pi}{3T}x - \pi$
  - R: $\frac{2\pi}{3T}$
- For $-\frac{T}{4} \leq x \leq \frac{T}{4}$
  - R: $\frac{10\pi}{3T}x - \pi$
  - G: $\frac{2\pi}{3T}$

These equations are easily translated into the C programming language to produce the stream of ideal positions.

Buehler Algorithm in Code

```
struct timeval buehler(int rpm, int* pos)
```

This is the prototype of the buehler function that produces the ideal positions to make the motors follow the Buehler motion. The function takes the desired revolutions per minute and the array of ideal position values for each leg as parameters and returns the current time for the use of the PD algorithm. First, the Buehler period is calculated using the rpm value. Then with the Buehler period, the boundaries defined in the Buehler motion graph so that the change in phase for the triplets can be performed. These variables are static, meaning they will retain their value after the function exits; this action saves substantially on computation time and every computation avoided is desired given that this function is extremely computationally heavy.

Using the gettimeofday function, the Raspberry Pi emulates a real time system and the timing is important in calculating the correct positions for the motors to be at. The start time is defined as a static variable and calculated only the first time the buehler function executes. Subsequent executions have the function get the current time and uses the difference between the current and start time to determine the elapsed time. This elapsed time is used to calculate the current phase position of the leg motors.

This current phase position is compared to the boundaries and based on the results of the comparisons, the ideal position for both triplets are calculated and returned to the calling function.

Follower Algorithm

Walk/Move Functions

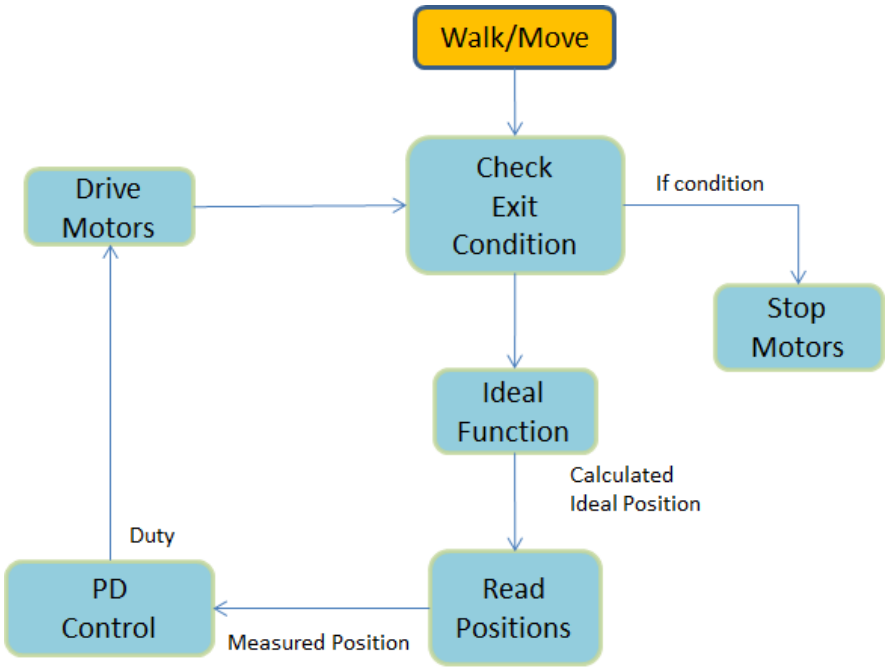


Figure 34 - Walk function structure.

As can be seen in Figure 34, the walk and move functions are extremely similar in their high level implementations. This is to be expected given their similarities; both were designed specifically to drive the leg motors as the operator requires. The walk function is very specific in its use, it drives the motors in the Buehler motion solely for the purpose of moving the rover from one place to another. The move function allows the legs to move to the leg motors to any arbitrary position. For this reason, both call the ideal position generator, read the motor positions, call PD control, and drive the motors. The two main reasons that the functions had to be separated are what ideal function is called and the exit condition for each function.

The ideal function generator for the walk function is Buehler while the move function makes use of the follower function. Regardless of the source of the ideal position value, both move and walk have to read motor positions and feed the measured and ideal positions into the PD control function to produce a calculated duty. This duty is then fed to the motors via the serial communication line.

Another difference is the exit condition for the functions. The walk functions counts the number of steps by comparing the last ideal position and the current ideal position to the starting position. When the current is larger than the starting and the last position is less than the starting position then a single step has been taken. Once the desired number of steps is reach, the rover stops walking. In contrast, the move function is passed an end position that all leg motors will be driven toward. A region is defined in proximity to the given end position. Once the ideal position is within the end position region, the function stops calculating ideal

positions and immediately drives each motor to the end position. With a small end position region, the motors are not strained by being quickly driven and stopped when then read the actual end position.

S. Communications

Modem

The modem that was chosen for 3G/4G wireless connection is the Novatel USB551L. The wireless technology that it supports is 4G LTE and CDMA. It is backwards compatible with 3G networks which will be useful in case of loss of 4G connection. The system requirements specify that a 166 MHz processor and 128 MB of RAM are required. The Raspberry Pi easily exceeds these requirements. The modem also supports the Linux operating system. Its dimensions in centimeters are 8.79 x 3.51 x 1.19 and it weighs 34.9 grams.

Router

The Router used to facilitate both communication between the operator and various components on the rover was the TP-Link TL-MR3430.This is a router that is compatible with various 3/4G modems to access the internet. The router then acts as a hub for us to connect our various devices to it via Ethernet. The router has 4 Ethernet ports available. We will use 2 Ethernet ports for the IP cameras and the other two will be used to provide internet connection for both of the Raspberry Pis to be able to communicate over the wireless network using SSH (Secure Shell). This allows us to remotely control the Raspberry Pis with the internet connection provided by the modem-router configuration. The router has a maximum power consumption of 8W when a 9 volt supply is used.

Cameras

The TP-Link TL-SC4171G is the pan and tilt internet protocol (IP) camera that will be mounted on the camera boom. It provides a 354° pan range and a 125° tilt range with a 50° viewing angle. This will give the operator a wide range of view to search for and identify the colored rocks during the competition. The camera also has a display resolution of 640x480 and an image frame rate of 25fps - 30fps. More importantly however, is the camera's ability to compress the video feed using the MPEG-4 video compression standard. That allows for remote viewing without the need of computation from the onboard computer, freeing clock cycles to be used for leg motor control. The camera also is equipped with an Ethernet port so that it can be connected into the greater communication network via the onboard router. The camera has a maximum power consumption of 12 W.

The second camera that will be used is the TP-Link TL-SC3230N IP camera. It is a fixed camera that will be mounted at the front of the rover for optimal view of the arm and gripper when picking up the colored rocks. It compresses video with the H.264 compression standard and will not require the onboard computer for video processing and transmission. Like the pan/tilt camera, it has an Ethernet port for connection into the communication network via the onboard router. Its maximum power consumption

is 3 W and its frame rate is 30fps with a 1280x1024 image resolution. Its maximum power consumption will be 12 W and its dimensions are 74.5 x 54.3 x 34.7 mm.

T. Graphical User Interface (GUI)

Purpose

The Graphical User Interface, or GUI, is a custom computer application which aims to greatly simplify the operation of the rover through integration of information display, in the form of video feeds and sensor data, and rover control. Actions performed by the interface can be separated into 6 categories: networking, video display, video processing, locomotion control, SEM control, and safeguarding.

Layout

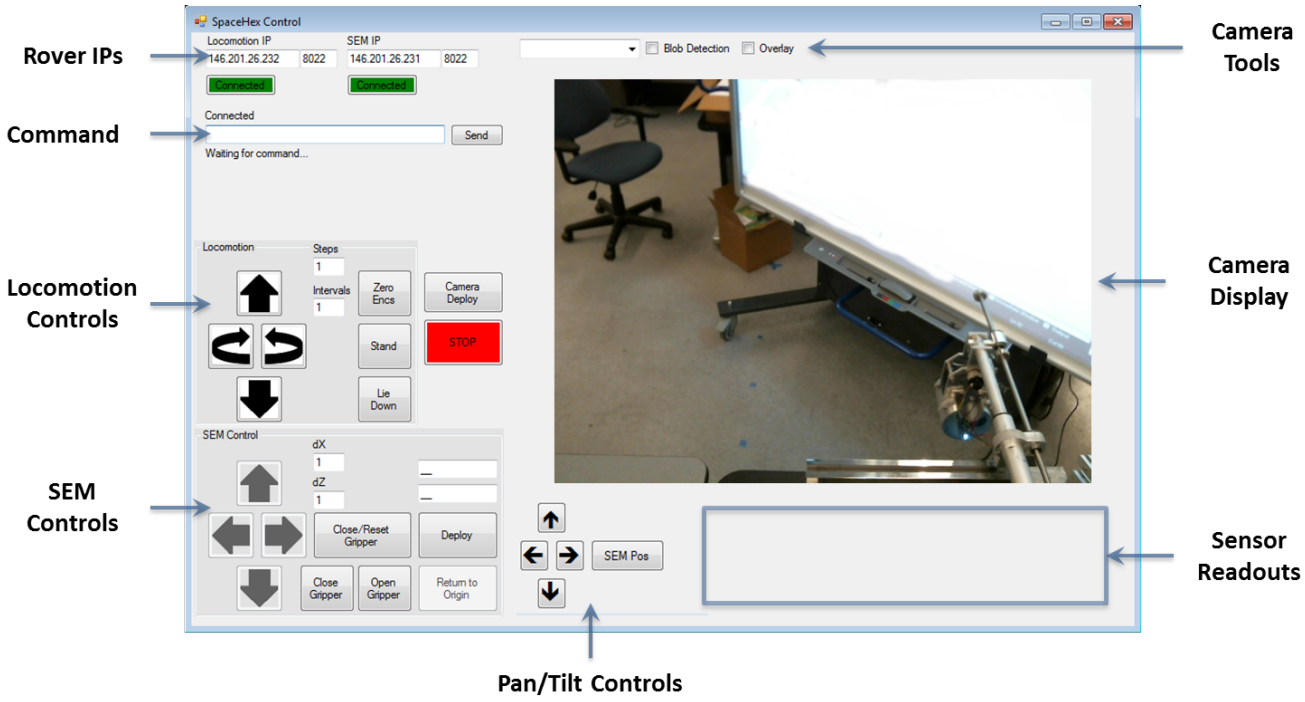


Figure 35 - GUI layout.

Networking

Complex networking procedures are required to establish communication between the cameras and computing systems on the rover and the control computer located at the college. The GUI has been programmed to handle these tasks automatically after initial setup, thus streamlining the control process and, ultimately, lowering sample extraction times. The figure below displays the required communication links required for rover operation.

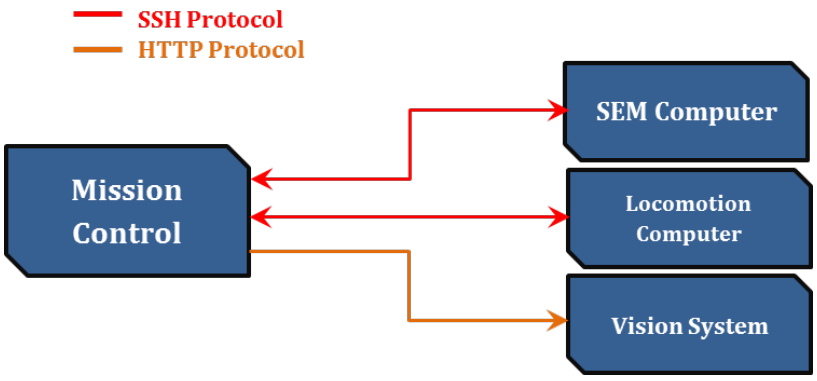


Figure 36 - Communication protocol requirements.

As the figure shows, communications via SSH (Secure Shell) must be established between the mission control computer and the on-board computers, and communication via HTTP is used to link the vision system (cameras) to the mission control computers. Ordinarily all of these communications links would be fairly straightforward to establish. In our case, both the mission control computer and networked hardware on the rover are behind NAT (Network Address Translation) firewalls. This particular type of firewall effectively prevents all incoming connections to devices behind the firewall. In our case this means that we must find a way to send commands and request video feeds through two firewalls which typically only allow outbound communication. Our solution to this problem involves adding an off-site server (not behind a NAT firewall) to the communication network and employing a combination of local and remote port forwarding.

First, SSH connections are established from the on-board computers to the server (performed automatically via a boot script), which allows incoming connections, and remote port forwarding (or "Reverse" SSH) is executed simultaneously. This links a port on each on-board computer to a port on the server. Local port forwarding is then used to link these server ports to ports on the router of the server's local network. Once this has been done, an SSH connection sent to these router ports by the mission control computer will be redirected to the appropriate ports on the server by the router, and then directly to the on-board computers by the server. The delay added by redirecting the communications through a server appears to be negligible. The server is also used to redirect the video feeds broadcast by the IP cameras from ports on the on-board network of the rover to ports on the mission control computer allowing the GUI to navigate to local ports to extract the video feeds from the cameras.

Despite the rather convoluted process of linking the control and on-board computing hardware, the result is simple: the user clicks a button on the GUI and the rover, possibly hundreds of miles away, responds appropriately.

Video Display and Processing

As stated above in the networking section the GUI is able to automatically extract the video feeds from the on-board IP cameras and insert them into the interface at the desired resolution and frame rate. The user is able to switch between camera feeds using a simple drop-down menu. Once the video feed is displayed, several options become available to optimize the effectiveness of the video feed.

Pan/Tilt Camera Control

Control of the orientation of the Pan/Tilt camera can be performed using the control built-in to the interface, shown below.

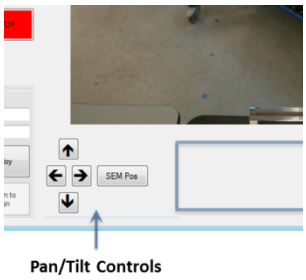


Figure 37 - Pan/Tilt Camera controls.

These controls allow the user to pitch the camera in the vertical and horizontal directions. The "SEM Position" button is used to return the camera to the pre-set orientation where "click-to-move" sample extraction is performed (this feature is described in the "SEM Controls" subsection).

Blob Detection

Blob detection is a video processing technique in which clusters of pixels within a specified RGB and physical size range are highlighted on the video stream. This technique, when utilized on the rover, can effectively increase the range of vision of the cameras. The algorithm was integrated into the GUI and can be activated by clicking the "Blob Detection" checkbox above the video feed. Shown be is an example of samples being highlighted by the algorithm.

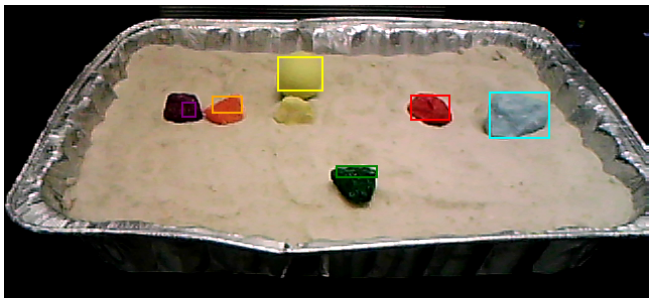


Figure 38 - An assortment of rock samples being highlighted by the blob detection algorithm.

The effectiveness of the blob detection algorithm depends on two main factors: the calibration (RGB ranges) and the quality of the video feed. The max/min RGB values can be manually adjusted to calibrate the algorithm, but this is a lengthy process, and if the lighting conditions change significantly the algorithm must be re-calibrated. To address this issue, a quick and easy calibration routine was developed and will be explained next.

When the "Blob Detection" checkbox is selected, a "Calibrate Filters" button appears. Pressing this button takes a screen shot of the current video feed and inserts it into a new window, shown below.

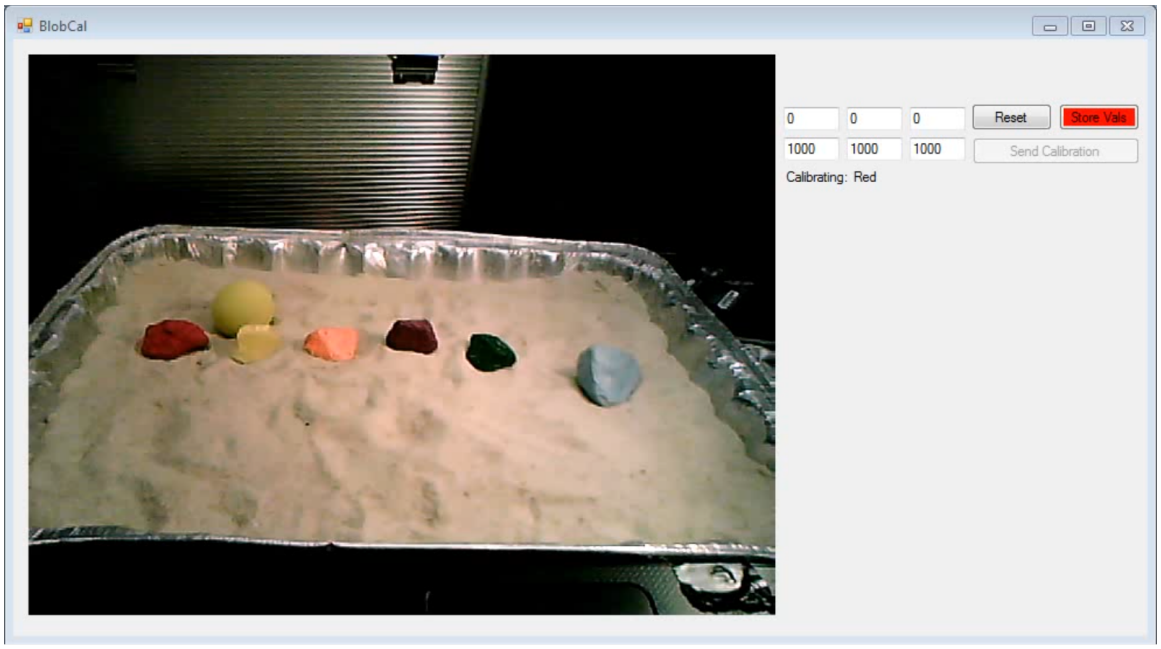


Figure 39 - Blob calibration window.

The calibration is performed by simply clicking on the sample indicated by the interface, red in the image above, 10 times in various locations. The interface stores the maximum red, green, and blue color values for all 10 times the sample was clicked. The interface then asks for the user to calibrate the next color, and so forth for all six colors. Once all colors are calibrated, the program prompts the user to return to the main interface. Once back in the main interface, the "Apply Calibration" button is clicked. If the user does not like the results of the calibration the routine can be repeated ad-infinitum.

As stated previously, the quality of the video feed is the second criterion for effective blob detection. If a high quality video feed could be streamed from the rover to mission control at a high frame rate, there would be no need for the blob detection algorithm, as human beings are much better at detecting blobs than this simple algorithm. Unfortunately, due to the limited bandwidth of the Verizon 3G connection, we are limited to extremely low-quality video feeds. It was found through research of video feeds from prior year's

competitions that it is not common for the maximum range that a colored sample can be distinguished from its surroundings, by eye or algorithmically, to be less than 15 feet.

Our solution to this problem is to pass a high-quality video feed to the blob detection algorithm at a low frame rate and to then superimpose the results produced by the algorithm onto the low-quality, higher frame rate pan/tilt camera. This process is depicted graphically in the figure below.

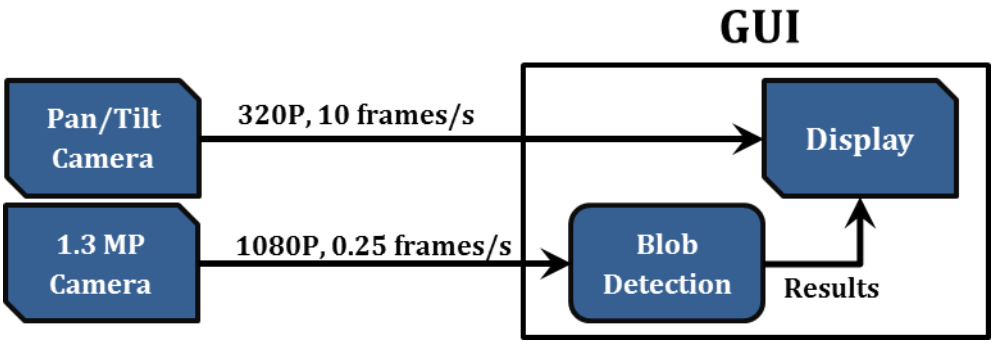


Figure 40 - Blob detection strategy.

With this strategy we are able to increase the distance at which we are able to detect samples dramatically, though the technique is subject to errors in the positions of the blobs due to differing perspectives of the cameras.

Locomotion Control

An important functional aspect of the GUI is the control of rover movement. Rover movement can generally be described by one of the following terms: standing, walking, turning, and lying down. Depending on the function, several parameters must be sent to the on-board computer along with the command itself, such as speed or number of steps. Due to the legged nature of the locomotion system, these parameters are integers, and are entered via general text fields. Once the proper supporting parameters have been entered, the user issues the command by pressing a button. The command is sent as a string to the thread which handles SSH connections, and is then sent to the locomotion computer on the rover. For example, a command could read as follows: "./rvr –w F 15 30". This would result in the rover walking forward for 15 steps at a leg speed of 30 RPM and would be initiated by entering these values into the corresponding fields and clicking the forward button. Turn commands are issued in the same manner. Some additional functions exist such as calibration of the legs (setting all decoders equal to zero at a common reference point), standing/laying down, and holding the legs at a specified orientation (such as 15,000 ticks) to control the height of the robot.

SEM Control

Two options exist for controlling the movement of the sample extraction module: manual control and click-to-move (CTM). Manual operation is performed by entering numerical values into the text fields which represent the desired movement in inches and then clicking the corresponding button to initiate movement.

Moves can be performed to displace the arm relative to its current position and to deploy the arm to an absolute position relative to the origin. Shown below are the interface controls used to perform manual control.

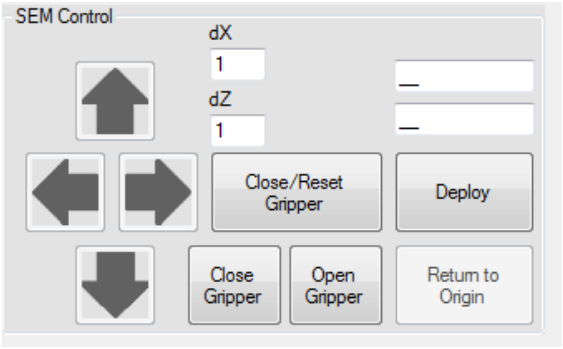


Figure 41 - Manual SEM controls.

Manual operation is initiated by entering numerical values into the fields above the deploy button, corresponding to absolute X and Z positions, and clicking deploy. Once the arm is deployed, relative moves are enabled (they are disabled in the figure above to prevent damage to the arm), and can be performed by entering a dx or dz (left/right or up/down) value in inches and clicking the corresponding button. When the sample is below the gripper, the close gripper button is clicked to capture the sample, and then the 'Return to origin" button is clicked to return the gripper to its original location above the storage bin. The sample is then dropped into the bin for storage.

The click-to-move method of control is simple in concept but required extensive programming to implement. To enable CTM, the camera must be in the "SEM Position" and the video overlay must be enabled which displays the extraction region of the SEM (shown below).

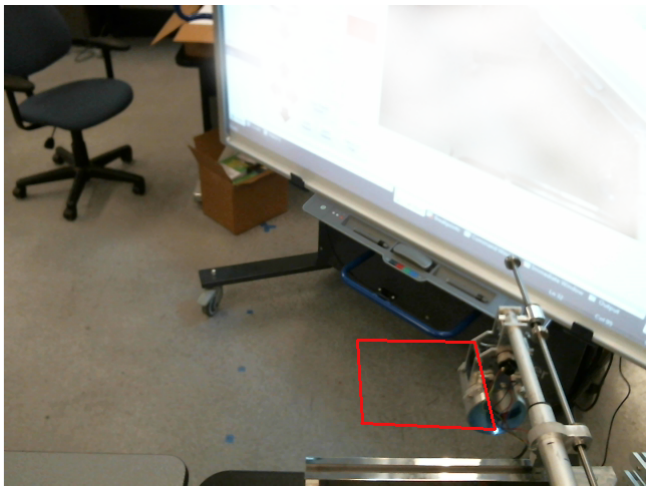


Figure 42 - Camera in sample extraction orientation with overlay to indicated extraction region.

Once these conditions have been met the CTM feature will be enabled automatically by the interface. If the arm has already been deployed, as shown above, holding down the right mouse button and left-clicking anywhere within the extraction zone will result in the gripper automatically moving to the point of the click. If the arm has not yet been deployed it will to deploy to the site of the click. Algorithmically, the interface is simply converting the horizontal and vertical locations of the pixel that is clicked on into the Cartesian space of the SEM. The algorithm takes the perspective of the camera into account. This method of control is only feasible because of the planar design of the arm and this control implementation should allow us to take full advantage of this design and to achieve low sample extraction times.

Multi-Threading

Multi-threading has been integrated into the GUI to increase its responsiveness and efficiency, to allow multiple commands to be sent simultaneously to the rover, and to establish a hierarchy which allows certain commands to take precedence or override others. As an example, the figure below depicts the differences between an SEM move performed in single-threaded and multi-threaded environments.

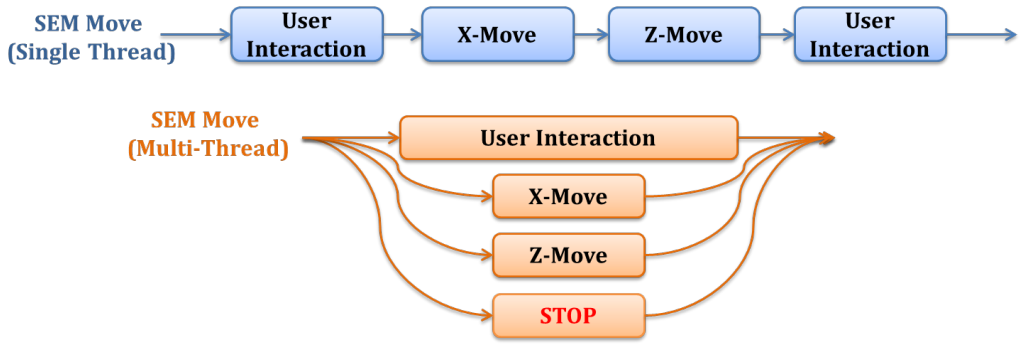


Figure 43 - Single and multi-threaded program flows.

As the figure shows, the x and z move commands can be sent simultaneously in the multi-threaded environment. This can reduce sample extraction time by as much as 50%, depending on the distances traveled. In addition, the multi-threaded environment allows the user to retain interface responsiveness throughout the process, meaning the GUI will not "lock up" while it is issuing the commands and waiting for a response from the rover. This also allows the user to issue an emergency stop command which will immediately trump all move commands being processed by the rover. In the single threaded environment this would not be possible and it would be left up to the on-board rover software to prevent damage to the arm if something went awry.

When the GUI is first started, 3 connections to each of the on-board computers are opened to allow multi-threading of commands, 2 of these connections are for general commands and the third is reserved for emergency stop commands. There are safeguards in place to prevent the same command being issued to the rover at the same time. For example, if the user accidentally clicks the "left" move button twice, the interface will prevent the second command from executing until the first has finished.

Safeguards

In addition to the safeguarding provided by emergency stop commands with dedicated threads as discussed above, the GUI offers an opportunity to include redundant safeguards for both the SEM and Locomotion systems. Although there are safeguards in place on the rover on-board software to prevent invalid SEM moves and to make sure the rover does not tip or flip, implementing these safeguards again in the GUI increases system reliability and the team's confidence in the rover.

U. Environmental and Safety Concerns

Kill switches will be used during testing to ensure that the robot can be safely switched off if any unexpected behavior is observed. Proper precautions will be observed during the wiring of the rover's electrical systems to minimize the risk of shock. Goggles will be worn when testing the rover in sand or other loose terrains as there is a high likelihood of debris being flung by the rover's legs. Lithium Ion batteries will be the source of electrical power on the rover, as such, a team member will be present at all times during the charging process as there exists a fire hazard with batteries of this type. Extra caution will be exercised at public outreach events, where children will be in close proximity to the rover.

V. Arm Concepts

Pulley Arm Concept

This arm concept design utilizes a pulley mechanism which moves a gripper along a track. Figure 8, below, shows the arm itself as a track which can be raised and lowered in reference to the ground, as well as pivoted about its base. The three degrees of freedom of the arm allows for a large reach area. When an object is picked up by the gripper, the arm can fold back over the robotic platform where it can release the object in to a collection-bin located on the top surface.

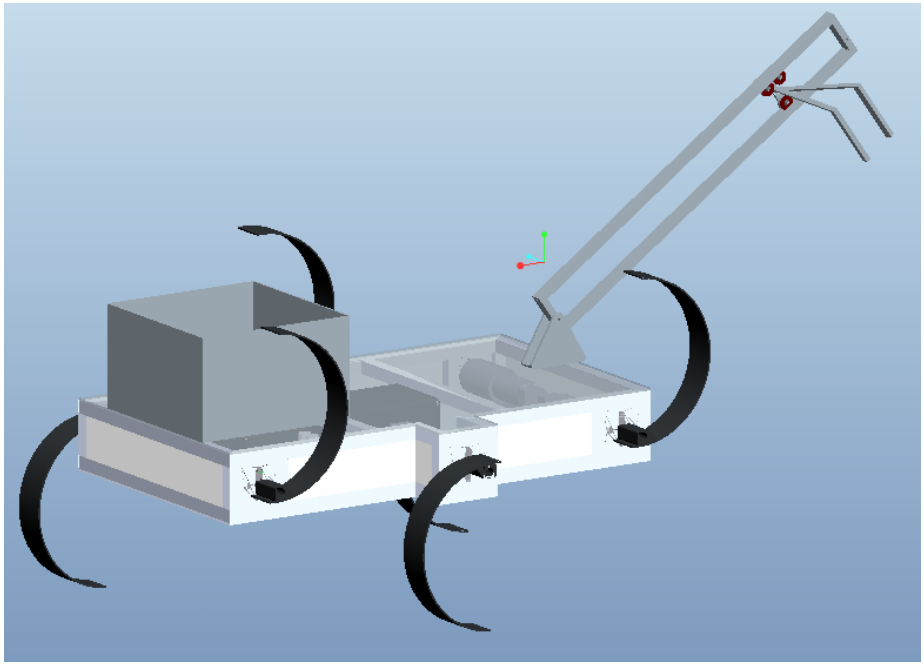


Figure 44: Pulley Arm Concept on the robotic platform with storage box.

One of the benefits of this design is that almost any gripper/scoop design can be implemented on the arm. This allows for modifications to gripper designs to be easily made if design problems arise. Also, the three plane operation allows for a wider range of gripper positions when reaching for an object as opposed to an arm which operates in two planes.

On the other hand, a downside to this design is that the pulley system is open to the elements. Debris can potentially become lodged in the track and seize motion of the gripper. This design also requires more complex control algorithms as opposed to an arm which operates in two planes.

Three Degree of Freedom Manipulator Concept

This robotic manipulator concept is a 3-DOF arm consisting of all revolute joints. It was evaluated that this is the minimum mobility necessary to manipulate specimens as required while maintaining versatility in how samples can be acquired. The joints are equivalent to a 2-DOF "shoulder", and a 1-DOF "elbow", with no "wrist" joint before the end-effector; servo or stepper motors may be used at each joint.

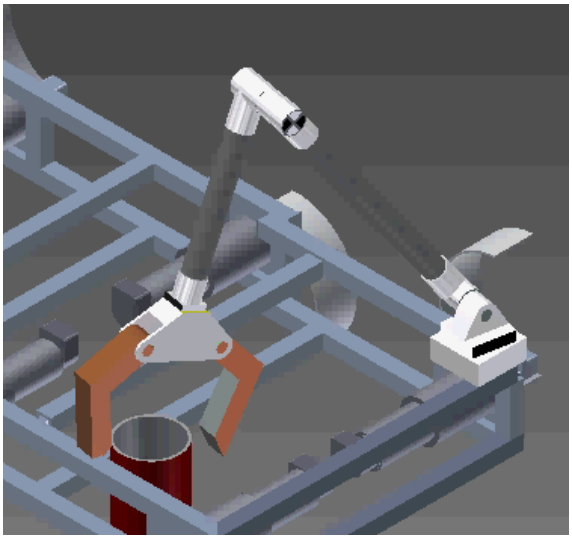


Figure 45: Robotic arm concept model generated using Autodesk Inventor Professional 2012. Gripper shown is generic and does not accurately represent gripper concepts generated.

Some key advantages of this concept is that its mobility allows access to a sample stowage compartment placed anywhere on the rover, and operate around any other instruments on top of the rover. Also, this arm can be mounted in front of or on top of the rover without jeopardizing its function. It is very compact in its stowed configuration, thus it easily meets the dimensional stowed rover configuration requirements.

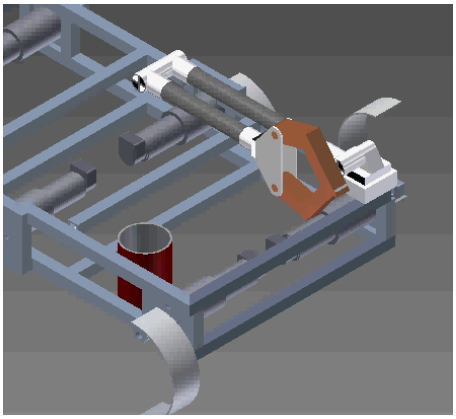


Figure 46: Robotic arm concept in stowed configuration. Model generated using Autodesk Inventor Professional 2012. Gripper shown is generic and does not accurately represent gripper concepts generated.

A major downside to this design is the complexity in controlling and/or automating the system. The number of motors that would be employed for the arm joints, and whatever gripper mechanism is implemented,

would require some relatively complex control algorithms to simplify the control scheme to something more user friendly.

Planar Arm Concept

This concept addresses the question: Is there a difference between a robotic arm designed for a wheeled robot and one designed for a legged robot?

A wheeled robot is "planar" in that it cannot adjust the vertical position of its body; on the other hand, it is clear that a legged robot is non-planar because it can adjust the height and even the angle of its body by manipulating the orientation of its legs. This arm concept takes advantage of this fact by removing the vertical degree of freedom generally found in robotic arms and instead utilizing the legs of the robot to adjust the vertical position and angle of the end effector. The result is an arm that is very simple to control, requiring only two linear motion axes. The figure below displays the axes of motion for this design (marked in red).

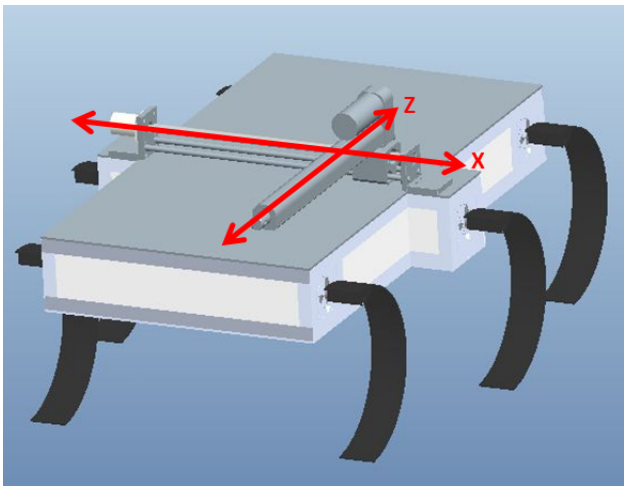


Figure 47: Depiction of the two axes of motion the Planar Arm Concept operates on.

The robot initiates the extraction process by lowering the plane of arm motion to the plane of the sample using the legs. In the above case, the sample (red ball), is located on flat ground so the robot simply lays all the way down, this is expected to be the case a majority of the time during the competition from examination of footage from previous years. The robot then positions the claw or end effector over the sample, captures it, and returns it to the storage container on the front of the robot. A scoop/pincer hybrid claw is shown above for demonstration purposes and is not the only possible claw configuration for use with this arm design. This sample extraction process is depicted in Figure 12 on the next page.

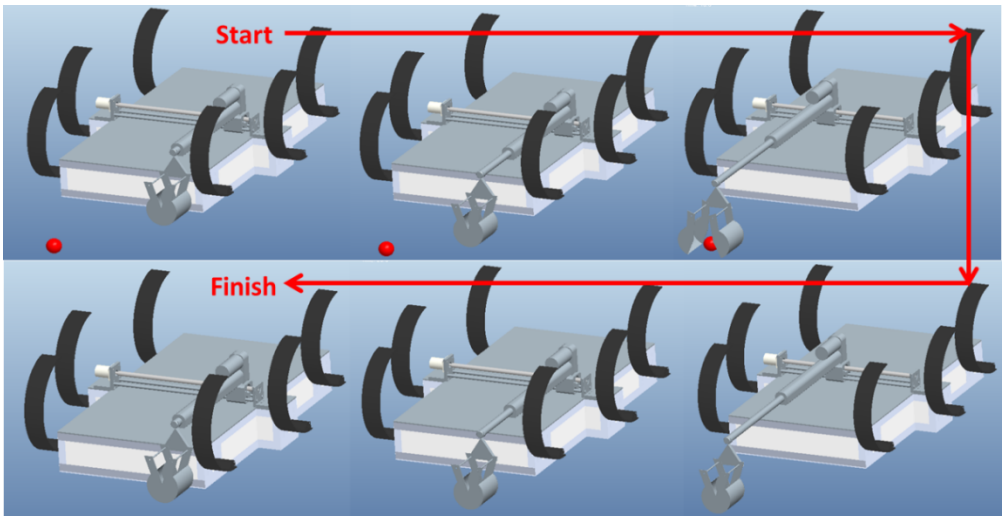


Figure 48: Step-by-step depiction of the Planar Arm Concept's sample extraction process.

Advantages

The operator will be controlling the robotic arm remotely and will guide the arm based on a low quality and low frame rate video, which will be delayed by several seconds. Every degree of freedom added to the arm will exponentially increase the time required to collect a sample with the arm, and as such, this arm has a significant advantage over the other proposed designs with respect to ease of control.

The number of motors or actuators required for the operation of an robotic manipulator is directly related to its weight, cost, and reliability. This design has the fewest number of required motors of the designs studied, thus it may have a slight edge in these categories (depending on specific hardware used). Also, the plane of the arm is located just above the top surface of the robot, which keeps the center of gravity low and results in a more stable robot. Finally, the end effector can be used to easily push away undesired rocks to isolate the target sample.

Shortcomings/issues

The design has several shortcomings and possible issues. The design requires that the storage box for the samples be mounted on the front of the robot, which would add to the length of the robot. As we are well under the maximum allowable dimensions specified by the competition, this is not an issue. Secondly, it is hard to predict how effective the legs will be in controlling the height of the robot, it may be the case that the robot is not stable at some intermediate positions between the prone and standing positions or that the resolution between these positions is too low for precise control. If this arm design is chosen, adequate control would be verified in the design of the electronics and a study would be performed to identify possible unstable positions.

W. Gripper Concepts

Pincer Style Claw

So called “pincer” style claws attempt to mimic the way relatively small objects are most commonly secured by the human hand. These claws generally consist of prongs or fingers which move towards each other to capture an object and prevent further motion through continuous application of force. This style of claw is good at picking up discreet objects but requires a relatively high level of precision from the manipulator it is attached to.

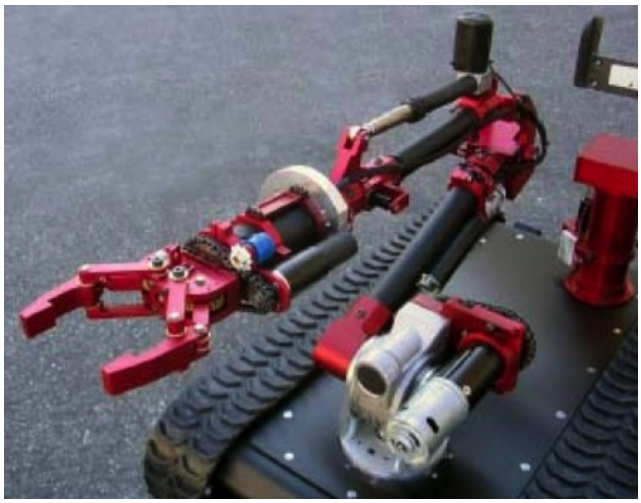


Figure 49: An example of a pincer-gripper. Image obtained from the Science in Seconds Blog.

Scoop Concept

There exist many versions of a “scoop” style sample/substance acquisition system, but they are all based on the idea of using the geometry (generally a concave surface) and the direction of gravity to capture and retain an object or substance. Scoops are generally used to pick up large quantities of a material and are not ideal for acquisition of discreet objects. Scoops can be operated successfully with much less precision than pincer style claws.

Figure 50: An excavator with the item of interest, its scoop, encircled.

Pincer/Scoop Hybrid

The scoop and pincer designs can be combined to form a claw that can pick up both discreetly and in bulk, is easy to control, and offers fairly high precision. A pincer/scoop hybrid claw moves two concave surfaces in a pinching motion to capture objects.

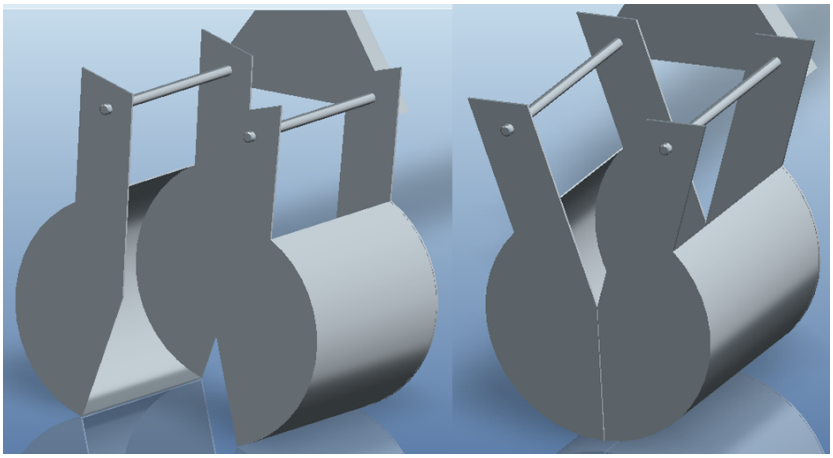


Figure 51: A solid model of the hybrid concept generated in Pro/Engineer software.

Universal Jamming Gripper Concept

This gripper concept utilizes not so common technique of picking up objects. Instead of having rigid moving members which grasp or scoop and object, this universal gripper conforms to the object in which it is grasping. The gripper consists of an ordinary latex party balloon filled with ground coffee. When the coffee-filled balloon is pressed onto the desired object to be picked up, the balloon and coffee conform to the object. At this point, a vacuum pump evacuates air from the balloon, solidifying the balloon, and thus gripping the object. This solidification is due to a "jamming transition" experienced by the coffee. When the air is vacated from the coffee filled balloon, the particulates of the coffee are pressed against each other causing them to resist slipping by one another or causing "jamming."

Figure 52: The universal gripper conforms to the shape of any object it is lifting to allow for a delicate yet firm grasp

This concept is very beneficial in that the gripper will not have to orient itself to the object being picked up, but rather simply press against it. Conventional grippers require the target object to be oriented a certain way between the contact points to be picked up. This concept has several flaws when it comes to implementation in the competition. Although the universal gripper excels at easily gripping objects, it will also grip objects adjacent to the target object. The vacuum seal of the gripper can be compromised by sharp objects which can puncture the latex balloon. Also, the lack of need for orientation to the target object results in the lack of ability to un-wedge objects from tight spaces.

X. Camera Concepts

Internet Protocol (IP) Camera

IP cameras are typically used for surveillance purposes. For this reason, they feature the ability to pan and tilt. They also feature standard video transmission capabilities that can be remotely viewed from any personal computer (PC). These features are perfectly suited to the purposes of the competition. The standard pan/tilt ability of the camera would allow the rover operator to have a wide visual range. Another advantage to this type of camera is that it does not require any type of computational device; it is a completely standalone device. That is, it handles the video compression and processing by itself and then transmits the data. These cameras are also specifically built for outdoor use, which is where the competition will take place.

These capabilities do not come without a price, though, as IP cameras of this caliber are more expensive than standard webcams. Configuration of the device is also more complicated in that special care needs to be taken to ensure that all of the network values are correct. If the networking between the computer and the camera is not done correctly, then communication between the two would be impossible.

Figure 53: A typical IP camera (left) and webcam (right).

Standard Web Camera (Webcam)

The webcam is almost the complete opposite of the IP camera. It is cheaper and uses an onboard computer to handle video processing. There is also no need to worry about the correct network configurations since the onboard computer would take care of the communication aspect. As mentioned, this device would need an

onboard computer for its data processing. This has the potential to consume more power than the standalone IP camera solution. The video streaming would also not be as straightforward since webcams are not set up to automatically stream as IP cameras are. Furthermore, webcams are not built for use outdoors. They are primarily social cameras for use with Skype and other software which, in most cases, does not involve the camera being used outdoors.

Y. Budget Overview

Purchase	Cost
Drive Motors	$6,099
Raw Materials	$352
Misc. Hardware	$265
Configured Hardware	$1,032
Electrical Hardware	$1,663
Verizon Service	$285
Donated Items (Retail)	$835
Batteries	$3500
Travel and Lodging	$2500
Total Spent	$16,531

Funding		
Sponsor	**Initial**	**Remaining**
AME	$500	$0
Misumi	$1,000	$313
CISCOR	$3,000	$0
NASA	$10,000	$227
Space Grant	$1,000	$0

Table 5 - Itemized expenditures (left). Project funding (right).

Z. Source Code

Buehler.h

```
/*****************************************************************************/
#ifndef BUEHLER_H
#define BUEHLER_H

#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>

#define OFFSET1 33852                                   //Offset for even triplet
#define OFFSET2 12415                                   //Offset for odd triplet
#define BOUND 50

//Caculates the ideal Position
struct timeval buehler(int rpm, char dir, int* pos)
{
        static struct timeval startTime = {.tv_sec = -1, .tv_usec = -1};
        struct timeval currTime;
        static int buehlerPeriod;
        static int T1,T2;
        int P1,P2;

        //Initialize function if its the first call in the loop
        if(startTime.tv_sec == -1)
        {
                //Get start time
                gettimeofday(&startTime,NULL);

                //Calculate buehler period
                buehlerPeriod = (int) (((float)(60)/rpm)*1000000);

                //Calculate transition points
                T1 = (int) ((float)(buehlerPeriod)/4);
                T2 = buehlerPeriod - T1;
        }

        //Get current time
        gettimeofday(&currTime, NULL);

        //Caculate the current buehler phasor
        unsigned long long buehlerPhase = (unsigned long long) ((currTime.tv_sec -
startTime.tv_sec)*1000000 +

                                        (currTime.tv_usec - startTime.tv_usec)) % buehlerPeriod;

        //printf("buehlerPeriod : %d, beuhlerPhase %d startTime %d currTime %d\n", buehlerPeriod,
buehlerPhase, startTime, currTimeInt);

        //Get positions of both buehler cycles
        if(buehlerPhase <= T1)
        {       P1 = ((int) (((float)(1)*NUM_POS*buehlerPhase)/(3*buehlerPeriod))) % NUM_POS;
```

```
                P2 = ((int) (((float)(5)*NUM_POS*buehlerPhase)/(3*buehlerPeriod))) % NUM_POS; }
        else if (buehlerPhase < T2)
        {       P1      =       ((int)      ((((float)(5)*NUM_POS*(buehlerPhase-T1))/(3*buehlerPeriod))      +
((float)(1)*NUM_POS)/12)) % NUM_POS;
                P2      =       ((int)      ((((float)(1)*NUM_POS*(buehlerPhase-T1))/(3*buehlerPeriod))      +
((float)(5)*NUM_POS)/12)) % NUM_POS; }
        else
        {       P1      =       ((int)      ((((float)(1)*NUM_POS*(buehlerPhase-T2))/(3*buehlerPeriod))      +
((float)(11)*NUM_POS)/12)) % NUM_POS;
                P2      =       ((int)      ((((float)(5)*NUM_POS*(buehlerPhase-T2))/(3*buehlerPeriod))      +
((float)(07)*NUM_POS)/12)) % NUM_POS; }

        //printf("buehlerPeriod: %d, buehlerPhase: %llu, T1: %d, T2: %d, iPos[0]: %d, iPos[1]:
%d\n",buehlerPeriod, buehlerPhase, T1, T2, pos[0], pos[1]);

        //Adjust position with offset and direction
        if(dir == 'B')
        {       pos[0] = pos[2] = pos[4] = NUM_POS - (P1 + (NUM_POS - OFFSET1)) % NUM_POS;
                pos[1] = pos[3] = pos[5] = NUM_POS - (P2 + (NUM_POS - OFFSET2)) % NUM_POS; }
        else if(dir == 'R')
        {       pos[0] = pos[2] = (P1 + OFFSET1) % NUM_POS;
                pos[1] = (P2 + OFFSET2) % NUM_POS;
                pos[3] = pos[5] = NUM_POS - (P2 + (NUM_POS - OFFSET2)) % NUM_POS;
                pos[4] = NUM_POS - (P1 + (NUM_POS - OFFSET1)) % NUM_POS;      }
        else if(dir == 'L')
        {       pos[0] = pos[2] = NUM_POS - (P1 + (NUM_POS - OFFSET1)) % NUM_POS;
          pos[1] = NUM_POS - (P2 + (NUM_POS - OFFSET2)) % NUM_POS;
                pos[3] = pos[5] = (P2 + OFFSET2) % NUM_POS;
                pos[4] = (P1 + OFFSET1) % NUM_POS; }
        else
        {       pos[0] = pos[2] = pos[4] = (P1 + OFFSET1) % NUM_POS;
                pos[1] = pos[3] = pos[5] = (P2 + OFFSET2) % NUM_POS; }

        return(currTime);
  }

  //Walking Algorithm
  void walk(int rpm, char dir, int numSteps)
  {
        struct timeval tv = {.tv_sec = 0, .tv_usec = 0};
        int ipos[6] = {0}, mpos[6] = {0}, duty[6] = {0}, done[6]={0};
        int stepNum = 0, lpos = 0;

        //Walk numSteps
        while(stepNum < numSteps)
        {
                tv = buehler(rpm,dir,ipos);
          readAllMotorPos(mpos);
                PD(&tv,ipos,mpos,duty);
                driveAllMotors('H',duty);
        if((dir=='L')||(dir=='B')){
                if(((OFFSET1 > (*ipos)) && (lpos > OFFSET1)))// || ((OFFSET2 < (*ipos+1)) && (lpos <=
OFFSET2)))
                        stepNum++;
                lpos = *ipos;
```

```
        }
        else{
                if(((OFFSET1 < (*ipos)) && (lpos < OFFSET1)))// || ((OFFSET2 < (*ipos+1)) && (lpos <=
OFFSET2)))
                        stepNum++;
                lpos = *ipos;
        }}
        //Stops motors as they get to the ideal stop position(Steady State Error)
        int i = 0,j = 0;
        while(i < NUM_MOTORS)
        {
                        readAllMotorPos(mpos);
                        for(j = 0; j < NUM_MOTORS; j++)
                                if((((mpos[j] > OFFSET1-250)&&(mpos[j] < OFFSET1))||((mpos[j] >
OFFSET2-250)&&(mpos[j] < OFFSET2)))&&(done[j]==0))                      //If mpos is at standing
position
                                {
                                        done[j]=1;
                                        stopMotor(j);
                                        i++;
                                }
        }
        //Ensure motors are stopped
        stopMotors();
        return;
  }

  //Ideal follower for moving legs angles less than 2*pi
  struct timeval follower(int rpm, char dir, char legs, int flag, int* spos, int* pos)
  {
        static struct timeval startTime = {.tv_sec = -1, .tv_usec = -1};
        struct timeval currTime;
        static int period;
        int cpos;

        //Get start time if not initialized
        if((startTime.tv_sec == -1) || (flag == 1))
        {
                gettimeofday(&startTime,NULL);
                period = (int) (((float)(60)/rpm)*1000000);
        }

        //Get current time and set all positions to ideal start
        gettimeofday(&currTime, NULL);

        //Calculate current phase position
        unsigned long long phase = (unsigned long long) ((currTime.tv_sec - startTime.tv_sec)*1000000 +


        (currTime.tv_usec - startTime.tv_usec)) % period;
        cpos = (int) ((((float)(1)*NUM_POS*phase)/(period))) % NUM_POS;

        //Calculate position with start offset and proper direction for all legs
        int i;
        for(i = 0; i < NUM_MOTORS; i++)
```

```
                if(dir == 'F')
                        pos[i] = (spos[i] + cpos) % NUM_POS;
                else
                        pos[i] = NUM_POS - (cpos + (NUM_POS - spos[i])) % NUM_POS;

        //If only 1 triplet is being driven zero the other triplet
        if(legs == 0)
                i = 1;
        else
                i = 0;

        if(legs != 'A')
                for(i; i < NUM_MOTORS; i+=2)
                        pos[i] = spos[i];

        return(currTime);
}

void move(int rpm, char dir, char legs, int epos)
{
        int ipos[6] = {0}, mpos[6] = {0}, spos[6] = {0}, duty[6] = {0}, done[6] = {0};
        struct timeval tv = {.tv_sec = 0, .tv_usec = 0};
        int numMotors = 0, numDone = 0, i = 0;

        if(legs    == 'A')
                numMotors = 6;
        else if((legs == 1) || (legs == 0))
                numMotors = 3;

        readAllMotorPos(spos);
        follower(rpm,dir,legs,1,spos,ipos);
        while(numDone < numMotors)
        {
                tv = follower(rpm,dir,legs,0,spos,ipos);
                readAllMotorPos(mpos);

   //Check  to see which motors have finished
                for(i = 0; i < NUM_MOTORS; i++)
                {
                        if((mpos[i] > epos - 500) && (mpos[i] < epos + 500))
                                if(done[i] == 0)
                                {
                                        done[i] = 1;
                                        numDone++;
                                }
                        if(done[i] == 1)
                                /*duty[i] = 0;*/ ipos[i] = epos;
                }
                //Drive Motors that havent finished
                PD(&tv,ipos,mpos,duty);
                driveAllMotors('H',duty);
        }
        //Ensure motors are stopped
        stopMotors();
}
```

```
void hold(int hpos)
{
	int iPos[6] = {hpos,hpos,hpos,hpos,hpos,hpos};

	struct timeval tv = {.tv_sec = 0, .tv_usec = 0};
	int mpos[6] = {0};
	int duty[6] = {0};

	while(1)
	{
		readAllMotorPos(mpos);
		PD(&tv,iPos,mpos,duty);
		driveAllMotors('H',duty);
		delay(25);
	}

}
#endif
/*****************************************************************************/
```

Motor.h

```
#ifndef MOTOR_H
#define MOTOR_H

//define directions, and size of SPI buffer
#define FORWARD 1
#define		REVERSE 0
#define SIZE 12
#define NUM_MOTORS 6
#define NUM_POS 42875
#define HPOS 21438

//Headers
#include <wiringPi.h>
#include <wiringPiSPI.h>
#include <wiringSerial.h>
#include <stdlib.h>

//Global SPI, file descriptor, and pin # variables
int SPI_CHAN = -1, SPI_SPD = -1, FD = -1, DEC_RST = -1, ESTOP = -1;

//Reads one motor position
int readMotorPos(int motorNum)
{
	int pos = -1;
	unsigned char readBuff[SIZE] = {0};

	//Send and read data, format into an integer
	wiringPiSPIDataRW(SPI_CHAN,readBuff,SIZE);
	pos = readBuff[2*motorNum] << 8 | readBuff[(2*motorNum)+1];

	return(pos);
}

//Drives one motor
void driveMotor(int motorNum, int dir, unsigned char duty)
{
	unsigned char addr, command, chksum;

	//Flush serial data buffer and delay
	serialFlush(FD);

	//Select correct motor driver
	if((motorNum == 0) || (motorNum == 3))
		addr = 128;
	else if((motorNum == 1) || (motorNum == 4))
		addr = 129;
	else
		addr = 130;

	//Select correct motor on motor driver
	if(motorNum <= 2)
		if(dir == FORWARD)
			command = 0x00;
```

```
                else
                        command = 0x01;
        else
                if(dir == FORWARD)
                        command = 0x04;
                else
                        command = 0x05;

        //Calculate checksum
        chksum = (addr + command + duty) & 0x7F;

        //Send Data
        serialPutchar(FD,addr);
        serialPutchar(FD,command);
        serialPutchar(FD,duty);
        serialPutchar(FD,chksum);
        delayMicroseconds(100);
        return;
}

//Reads all motor positions
void readAllMotorPos(int *pos)
{
        unsigned char readBuff[SIZE] = {0};

        //Send and read data, format into an integer
        wiringPiSPIDataRW(SPI_CHAN,readBuff,SIZE);

        int i;
        for(i = 0; i < NUM_MOTORS; i++)
        {
                pos[i] = 0;
                pos[i] = readBuff[2*i] << 8 | readBuff[(2*i)+1];
        }
        return;
}

//Drives all motors
void driveAllMotors(char dir, int *speed)
{
        unsigned char addr, command, chksum, duty;

        //Flush serial data buffer
        serialFlush(FD);

        int i;
        for(i = 0; i < NUM_MOTORS; i++)
        {
                //Select correct motor driver
                if((i == 0) || (i == 3))
                        addr = 128;
                else if((i == 1) || (i == 4))
                        addr = 129;
                else
                        addr = 130;
```

```
//Select correct motor command check directions and speed
if(dir == 'F')
{
	if(i <= 2)
		command = 0x00;
	else
		command = 0x04;

	if(speed[i] < 0)
		duty = 0;
	else
		duty = (unsigned char) speed[i];
}
else if (dir == 'B')
{
	if (i <= 2)
		command = 0x01;
	else
		command = 0x05;

	if(speed[i] > 0)
		duty = 0;
	else
		duty = (unsigned char) abs(speed[i]);
}
else if (dir == 'R')
{
	if (i <= 2)
		command = 0x00;
	else
		command = 0x05;

		duty = (unsigned char) abs(speed[i]);
}
else if(dir == 'L')
{
	if (i <= 2)
		command = 0x01;
	else
		command = 0x04;

	duty = (unsigned char) abs(speed[i]);
}
else if (dir == 'H')
{
	if(i <= 2)
		if(speed[i] < 0)
			command = 0x01;
		else
			command = 0x00;
	else
		if(speed[i] < 0)
			command = 0x05;
		else
```

```
                                        command = 0x04;

                        duty = (unsigned char) abs(speed[i]);
                }       else;

                //Calculate checksum
                chksum = (addr + command + duty) & 0x7F;

                //Send Data
                serialPutchar(FD,addr);
                serialPutchar(FD,command);
                serialPutchar(FD,duty);
                serialPutchar(FD,chksum);

                delayMicroseconds(100); //Adjust delay get as low as possible
        }

        return;
}

void stopMotors()
{
        int duty[6] = {0x00};
        driveAllMotors('F',duty);
        return;
}
void PD(struct timeval *ts, int *ipos, int *mpos, int *duty)
{
        static struct timeval lts = {.tv_sec = 0, .tv_usec = 0};
        static int lastErrorPos[6] = {0};
        float derivative[6], volt[6],THE_D = 100,THE_P = 0.01;
        int errorPos[6], dt;            //THE_P = 0.0075

        //Dt is 1 on the first run of PD but is calculated afterwards
        if ( ts->tv_sec == 0)
                THE_D = 0;
        else if(lts.tv_sec != 0)
                dt = (ts->tv_sec - lts.tv_sec)*1000000 + (ts->tv_usec - lts.tv_usec);
        else
                dt = 1;

        //Save last time stamp
        lts = *ts;

        //Control loop iterates for each leg
        int i;
        for(i = 0; i < NUM_MOTORS; i++)
        {
                if((ipos[i] < 2000) && (mpos[i] > 40875))
                        errorPos[i] = ipos[i] + (NUM_POS - mpos[i]);
                else if ((mpos[i] < 2000) && (ipos[i] > 40875))
                        errorPos[i] = -1*(mpos[i] + (NUM_POS - ipos[i]));
                else
                        errorPos[i] = ipos[i] - mpos[i];
```

```
                if(errorPos[i]>8000){
                        stopMotors();
                        exit(1);}

        //Porportional derivative algorithm
                derivative[i] = (errorPos[i] - lastErrorPos[i])/((float)(dt));
                volt[i] = THE_P*errorPos[i] + THE_D*derivative[i];
                lastErrorPos[i] = errorPos[i];
                duty[i] = (int) ((volt[i]/24.0)*127);

                //Bounds check
                if(duty[i] > 127)
                        duty[i] = 127;
                if(duty[i] < -127)
                        duty[i] = -127;
        }
  if(derivative[3] >250)
        printf("**************************************************************$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$##@@##@#@#!@%$^$^&%$&@%$&%$^&@#$%\n");
  int k = 3;
        printf("iPos[i]: %d, mPos[i]: %d, ePos[i]: %d, eSpeed: %f, Volt: %f, Duty[i]: %d\n",
ipos[k],mpos[k],errorPos[k],derivative[k],volt[k],duty[k]);
        return;
  }

  //Resets all the decoders to 0
  void resetAllDecoders()
  {
        //Set high for ~50 ms
        digitalWrite(DEC_RST,HIGH);
        delay(50);
        digitalWrite(DEC_RST,LOW);
        return;
  }

  void stopMotor(int motorNum)
  {
        unsigned char duty = 0x00;
        driveMotor(motorNum,FORWARD,duty);
        return;
  }
  /*
  void stopMotors()
  {
        int duty[6] = {0x00};
        driveAllMotors('F',duty);
        return;
  }*/

  void emergencyStop()
  {
        //Set high for ~50 ms
        digitalWrite(ESTOP,HIGH);
        delay(50);
        digitalWrite(ESTOP,LOW);
```

```
        return;
}

int Init(int SPIChannel, int SPISpeed, int decRstPin, int eStopPin)
{
        //Set header file variables
        SPI_CHAN = SPIChannel;
        SPI_SPD = SPISpeed;
        DEC_RST = decRstPin;
        ESTOP = eStopPin;

        //Setup calls for WiringPi Library, SPI and UART
        wiringPiSetup();
        if((wiringPiSPISetup(SPI_CHAN, SPI_SPD) < 0) ||
                ((FD = serialOpen("/dev/ttyAMA0",115200)) < 0))
                return(-1);

        //Set pins as output
        pinMode(DEC_RST, OUTPUT);
        pinMode(ESTOP,OUTPUT);

        //Write default pin values disable motors
        digitalWrite(DEC_RST,LOW);
        digitalWrite(ESTOP,HIGH);

        //Flush SerialBuffer
        serialFlush(FD);

        return(0);
}

#endif
/*****************************************************************************/
```

Main.c

```
/*****************************************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "motor.h"
#include "buehler.h"

void printHelp();

int main(int argc, char* argv[])
{
	struct timeval tv = {.tv_sec = 0, .tv_usec = 0};
	int ipos[6] = {0}, mpos[6] = {0}, duty[6] = {0x00};
	static unsigned char cal=0;

	//Initialize SPI,UART,DecoderRst pin,EmergencyStop pin
	Init(0,8000000,6,5);

	if(argc == 1)
	{
		printHelp();
		return(0);
	}
	else
	{
		switch(*(++argv[1]))
		{
			case 'w': printf("WALK\n");
				move(10,'H',1,OFFSET2);
				walk(40,'F',10);
				move(10,'F','A',OFFSET1);
				hold(OFFSET1);

				break;
			case 't': printf("TURN\n");
				move(10,'H',1,OFFSET2);
				walk(10,'R',2);
				move(10,'F','A',OFFSET1);
				hold(OFFSET1);
				break;
			case 'c': printf("CALIBRATE\n");

				duty[0] =
duty[1]=duty[2]=duty[3]=duty[4]=duty[5] = -3;
				driveAllMotors('B',duty);
				delay(8000);

				duty[0] =
duty[1]=duty[2]=duty[3]=duty[4]=duty[5] = 0;
				driveAllMotors('F',duty);
				break;
			case 'u': printf("STAND\n");
				move(10,'F','A',OFFSET1);
```

```
                                                        hold(OFFSET1);
                                                        break;
                        case 'l': printf("LIE\n");
                                                        move(10,'F','A',OFFSET1);

                                                        move(5,'F','A',OFFSET2);

                                                        break;
                        case 's': printf("ESTOP\n");
                                                        stopMotors();
                                                        break;
                        case 'r': printf("RESETDEC\n");
                                                        resetAllDecoders();
                                                        break;
                        case 'h': printf("HOLD\n");
                                                        break;
                        case 'm': printf("POSITION\n");
                                                        readAllMotorPos(mpos);
                                                        printf("mpos0: %d, mpos1: %d, mpos2:
%d, mpos3: %d. mpos4: %d, mpos5: %d\n", mpos[0],mpos[1],mpos[2],mpos[3],mpos[4],mpos[5]);
                                                        break;
                        case 'x': printf("TEST\n");
                                                        //walk(10,'F',3);
                                                        duty[0]                         =
duty[1]=duty[2]=duty[3]=duty[5] = 0;
                                                        duty[3] = 40;
                                                        driveAllMotors('F',duty);
                                                        delay(4000);

                                                        duty[0]                         =
duty[1]=duty[2]=duty[3]=duty[4] = 0;
                                                        duty[5] = 40;

                                                        driveAllMotors('F',duty);
                                                        delay(4000);
                                                        duty[0]                         =
duty[1]=duty[2]=duty[3]=duty[4]=duty[5] = 0;
                                                        driveAllMotors('F',duty);

                                                        break;
                        default : printf("**Invalid option**\n");
                                                        break;
                }
        }
        return 0;
}

void printHelp()
{
                printf("\t\t\t\nProgram Options:\n\n");
                printf("\t-w : Walk ex. 'rvr -w D X' where D={F,B} and X is num steps.\n");
                printf("\t-t : Turn ex. 'rvr -t D X' where D={L,R} and X is num steps.\n");
                printf("\t-c : Calibrate ex. 'rvr -c' resets the decoders to zero.\n");
                printf("\t-s : Stand ex. 'rvr -s' moves legs to standing position.\n");
                printf("\t-l : Lie Down ex. 'rvr -l' sets the rover down.\n");
```

```
                printf("\t-e : Emegency Stop ex. 'rvr -e' stops all motors!\n");
                printf("\t-b : Camera Boom ex. 'rvr -b' raises/lowers the camera mast.\n");
                printf("\n");
                return;
}
/*****************************************************************************/
```

Spi.vhd

```
-- Commented Out the recieve registers and process statements.
-- (Uncomment if you need to recieve from SPI bus)
----------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
----------------------------------------------------------------------------
entity spi is
	port(	CLK : in std_logic;
			SCK : in std_logic;
			SEL : in std_logic;
			-- MOSI: in std_logic;
			MISO: out std_logic;
			--RDY : out std_logic;
			TRAN: in std_logic_vector (95 downto 0));
			--RECV: out std_logic_vector (95 downto 0);
end entity;
----------------------------------------------------------------------------
architecture spi_arch of spi is

signal SCKreg, SELreg : std_logic_vector (3 downto 0); -- MOSIreg
signal bit_cnt : unsigned (6 downto 0) := "0000000";
signal SCKr, SCKf, SELf, SELa : std_logic; -- MOSId
--signal rx  : std_logic_vector (95 downto 0);
signal tx  : std_logic_vector (95 downto 0);

begin

	--SCKr  <= '1' when (SCKreg(3 downto 0) = "0011") and (SELa = '1') else '0';
	SCKf  <= '1' when (SCKreg(3 downto 0) = "1100") and (SELa = '1') else '0';
	SELf  <= '1' when (SELreg(3 downto 0) = "1100") else '0';
	SELa  <= '1' when (SELreg(1 downto 0) = "00") else '0';
	-- MOSId <= '1' when (MOSIreg(1) = '1') else '0';
	MISO  <= tx(95) when (SELa = '1') else '0';

SYNC:
	process(CLK)
	begin
		if(rising_edge(CLK)) then
			SCKreg  <= SCKreg(2 downto 0) & SCK;
			SELreg  <= SELreg(2 downto 0) & SEL;
			-- MOSIreg <= MOSIreg(2 downto 0) & MOSI;
		end if;
	end process;

CNT:
	process(CLK)
	begin
		if (rising_edge(CLK)) then
			if (SELa = '0') then
				bit_cnt <= "0000000";
			elsif (SCKf = '1') then -- and (SELa = '1') assumed
				bit_cnt <= bit_cnt + 1;
```

```
                        end if;
                end if;
        end process;

--RECV_REG:
--      process(CLK, bit_cnt)
--      begin
--              if((rising_edge(CLK)) and (bit_cnt = 95)) then
--                      RECV <= rx;
--              end if;
--      end process;
--
--RY:
--      process(CLK,bit_cnt)
--      begin
--              if(rising_edge(CLK)) then
--                      if      (bit_cnt = 95) then
--                              RDY <= '1';
--                      else
--                              RDY <= '0';
--                      end if;
--              end if;
--      end process;

--RxD:
--      process(CLK)
--      begin
--              if (rising_edge(CLK)) then
--                      if (SCKr = '1') and (bit_cnt <= 95) then --(SELa = '1') and
--                              rx <= rx(94 downto 0) & MOSId;
--                      end if;
--              end if;
--      end process;

TxD:
        process(CLK)
        begin
                if (rising_edge(CLK)) then
                        if (SELf = '1') then
                                tx <= TRAN;
                        elsif (SCKf = '1') and (bit_cnt <= 95) then -- (SELa = '1') and
                                tx <= tx(94 downto 0) & '0';
                        end if;
                end if;
        end process;

end architecture;
--------------------------------------------------------------------------------
```

quadratureDecoder.vhd

```
--------------------------------------------------------------------------------
library IEEE;
use ieee.std_logic_1164.all;
--------------------------------------------------------------------------------
entity d_ff is
	port(	D	: in  std_logic;
		Q	: out std_logic;
		CLK : in std_logic);
end entity;
--------------------------------------------------------------------------------
architecture d_ff_arch of d_ff is
begin
	process(CLK)
	begin
		if(rising_edge(CLK)) then
			Q <= D;
		end if;
	end process;
end architecture;
--------------------------------------------------------------------------------
library IEEE;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
--------------------------------------------------------------------------------
entity counter is
	port(	CLK	: in std_logic;
		RST : in std_logic;
		EN  : in std_logic;
		DIR : in std_logic;
		CNT : out std_logic_vector (15 downto 0));
end entity;
--------------------------------------------------------------------------------
architecture counter_arch of counter is
begin
	-- Cannot run process above 80Mhz
	process(CLK,EN)
	variable count : unsigned (16 downto 0) := "00000000000000000";
	begin
		if (rising_edge(CLK)) then
			if (RST = '1') then
				count := "00000000000000000";
			else
				if (EN = '1')then
					if (DIR = '1') then
						if (count < 85749) then
							count := count + 1;
						else
							count := "00000000000000000";
						end if;
					else
						if (DIR = '0') then
							if (count > 0) then
								count := count - 1;
```

```
                                                        else
                                                                count   :=      "10100111011110101";   --
85749
                                                        end if;
                                                end if;
                                        end if;
                                end if;
                        end if;
                        CNT <= std_logic_vector(count(16 downto 1));
                end if;
        end process;

  end architecture;
  ----------------------------------------------------------------------------------
  library IEEE;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;
  ----------------------------------------------------------------------------------
  entity quadratureDecoder is
        port(     A         : in std_logic;
                  B         : in std_logic;
                  CLK       : in std_logic;
                  RST : in std_logic;
                  OUTPUT            : out std_logic_vector (15 downto 0));
  end entity;
  ----------------------------------------------------------------------------------
  architecture quadratureDecoder_arch of quadratureDecoder is
  signal A_0,A_1,B_0,B_1,B_2,A_2 : std_logic;
  signal COUNT_EN, COUNT_DIR, FRST : std_logic;
  signal RSTreg : std_logic_vector (3 downto 0) := "0000";

  component d_ff
        port(     D         : in std_logic;
                  Q         : out std_logic;
                  CLK       : in std_logic);
  end component;

  component counter
        port(     CLK       : in std_logic;
                  RST : in std_logic;
                  EN  : in std_logic;
                  DIR : in std_logic;
                  CNT : out std_logic_vector (15 downto 0));
  end component;

  begin

        FRST <= '1' when RSTreg (3 downto 0) = "1111" else '0';

        DFF0: d_ff port map (A,A_0,CLK);
        DFF1: d_ff port map (A_0,A_1,CLK);
        DFF2: d_ff port map (A_1,A_2,CLK);
        DFF3: d_ff port map (B,B_0,CLK);
        DFF4: d_ff port map (B_0,B_1,CLK);
        DFF5: d_ff port map (B_1,B_2,CLK);
```

```
	CNT0 : counter port map (CLK,FRST,COUNT_EN,COUNT_DIR,OUTPUT);

	SYNC:
	process(CLK)
	begin
		if(rising_edge(CLK)) then
			RSTreg  <= RSTreg(2 downto 0) & RST;
		end if;
	end process;

	process(CLK)
	begin
		if(rising_edge(CLK)) then
			COUNT_DIR <= A_1 xor B_2;
			COUNT_EN <= (A_1 xor B_1 xor A_2 xor B_2);
		end if;
	end process;

end architecture;
--------------------------------------------------------------------------------
```