April 2019

ROVE: Robotics Mining Platform

Anh Hong Phuong Dao
Worcester Polytechnic Institute

Luis Alexander Delatorre
Worcester Polytechnic Institute

R.O.V.E: Robotics Mining Platform

A Major Qualifying Project

Submitted to the Faculty of

Worcester Polytechnic Institute

in partial fulfillment of the requirements for the

Degree in Bachelor of Science

in

Mechanical Engineering & Computer Science

4/25/2019

TEAM MEMBERS

Anh Dao, CS

Luis Delatorre, ME

ADVISORS

Professor Michael Ciaraldi

Professor Kenneth Stafford

Abstract

As Earth’s natural resources are running out, scientists are searching for possible substitutions for our planet and how to study these candidates. Robotic Mining Competition is NASA’s annual event for university students to design a robot that can undergo the mining challenge in the Mars-simulated landscape. Our project surrounded the WPI robot that participated in NASA’s Robotic Mining Competition and focused on modifying current design for the upcoming 2019 competition, while adhering to the new rules. The previous iteration of the robot, Ibex, suffered from jamming issues and was unable to excavate the regolith simulant utilized in the competition, in addition to the lack of competent self-localization. To address these issues, our team focused on understanding the underlying causes of these problems to mitigate the effects. We found that the previous manufacturing of the digging scoops and dated electronic hardware contribute towards negatively impacting the robot’s efficiency. We designed chain guides and altered the scoop fastening system to resolve the jamming and digging ineffectiveness. To install a stable autonomous mode, we swapped out the complex control system to a singular unified hardware. With these modifications, the robot is ready to compete in the 2019 Robotic Mining Competition.

Acknowledgement

This project would not have been possible without help from certain individuals that assisted us by giving us guidance or by supplying us with resources.

First, we would like to express our gratitude to our advisors Professor Michael Ciaraldi and Professor Kenneth Stafford, for their constant assistance in developing R.O.V.E and this report.

We would also like to thank Thomas Kouttron and Michael Cooke for the use of space in the Higgins Labs, in addition to manufacturing.

Finally, we would like to thank those who sponsored our project: the Computer Science and Mechanical Engineering Department of WPI, the office of the Dean of Arts and Science and the Dean of Engineering.

Table of Contents

List of Figures

List of Tables

Chapter 1: Background

1.1 NASA Robotic Mining Competition

1.1.1 The competition

Robotics Mining Competition (RMC), designed by NASA for college students to participate in, is a simulation challenge surrounding the real experience of excavated material known as regolith on Mars. The simulation is designed to replicate this mission, allowing university teams to compete and present new designs that may possibly be utilized in future space missions. The challenge is performed by a robot via remote control of an off-site base, simulating the real world experience of piloting the space rovers - such as the Mars Curiosity rover or the first rover on Mars, Sojourner [1].

Worcester Polytechnic Institute (WPI) has been participating in this competition with the same robot under new improvements for the last three years while winning the Regolith Mechanics award for the robot Markhor during the 2017 cycle. Last year's robot Ibex, performed well, however not to the extent of the excellence that WPI wishes to achieve.

1.1.2 Markhor

Markhor was the second robot from WPI that participated in NASA RMC after Moonraker in 2009. Markhor won Regolith Mechanics Award as the result of the ability to identify a specific regolith mechanics problem and improve their design to cope with it. Markhor drive was designed with a tread to drive over any obstacles in the arena, collection system was based off the bucket ladder design and dumping system was designed to hold a total capacity of 100kg. At the competition, Markhor was able to dig deep to 30cm to get to the gravel. Markhor's design focused heavily on the mechanical side which led to some electrical and software issues at the competition. In addition, due to the complex design of the collector and bucket and the aluminum frame, the robot was a little overweight.

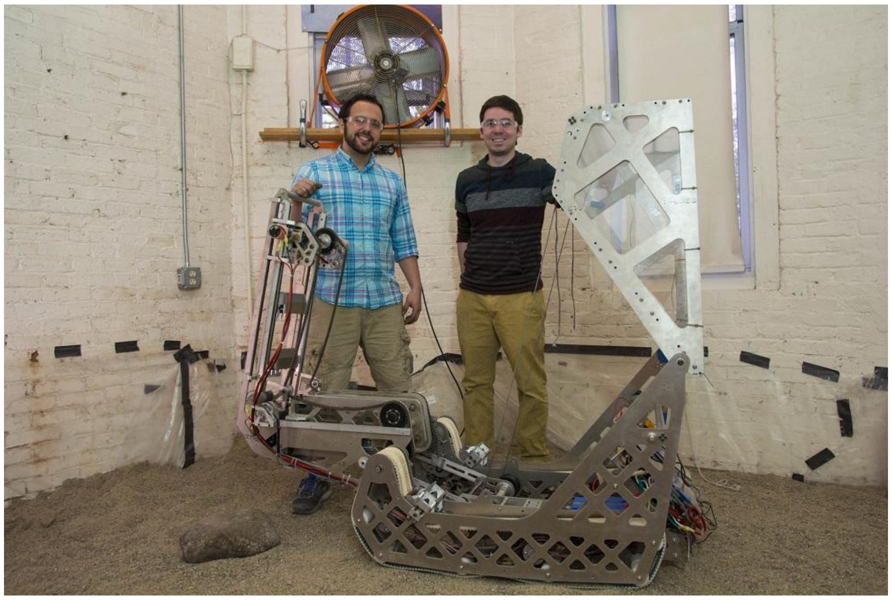


Figure 1: Markhor Design Iteration

1.1.3 Ibex

Last year's robot for WPI performed at the rank of 34 out of 44 robots that entered the competition. Ibex was able to maneuver to the excavating location and back to the dumping station, without slowing down from obstacles. In addition, the robot was able to locate itself in the arena using cameras, and utilizing its position, traverse the arena. The primary challenge Ibex had faced was during the improvements of the robots, the excavating system itself needed to be updated rather quickly. Because of this rush of design, the manufacturing and actual design weren't optimized. This caused Ibex to underperform on the digging part of the challenge. The motors didn’t have the required torque to run the chain system of the robot. This was understood after the competition last year in May 2018, but released as part of the report of the MQP involving Ibex.

1.2 Mechanical Design

1.2.1 Other digging designs

In the top-ranking teams of the Robotic Mining Competition of 2018, stands the University of Alabama, North Dakota State University in collaboration with James Madison University, and Kent State University. Comparing and contrasting the design of their excavation mechanisms can prove effective for optimizing the design of Ibex’s excavation system.

The team in first place in the competition is the University of Alabama with the team Astrobots. Their robot design for the year 2018 is not dissimilar to their robot during the 2017 cycle, which also had won first place during that competition as well. Both mechanisms used are bucket dredging systems that carry the regolith from the ground, up the chain system, and dropped over the dump bucket. [2] This system proved effective for two separate robots and competition cycles.

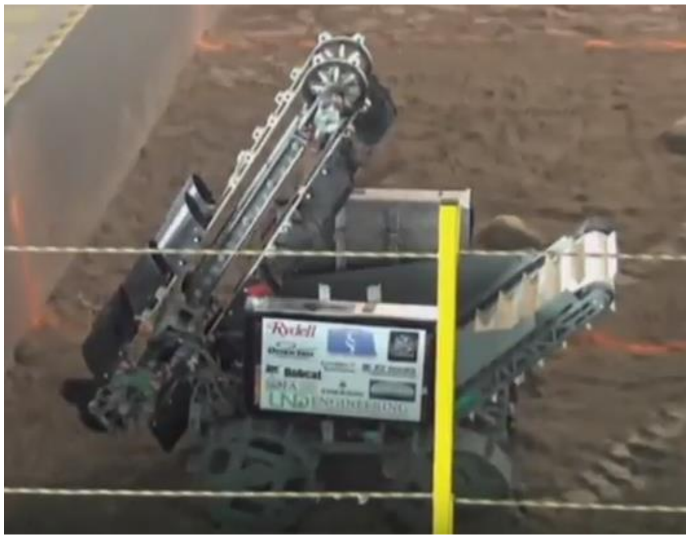


Figure 2: University of Alabama's Robot During Competition

North Dakota State University designed their robot in conjunction with James Madison University that reached second place in the competition. This used a scoop similar to that of a backhoe in construction vehicles. This excavation system was optimized to dig within the area of the chassis of the robot, however, to dump the regolith onto the robot itself to score points. With a background in digging already, this system is an obvious choice for optimal excavating design.

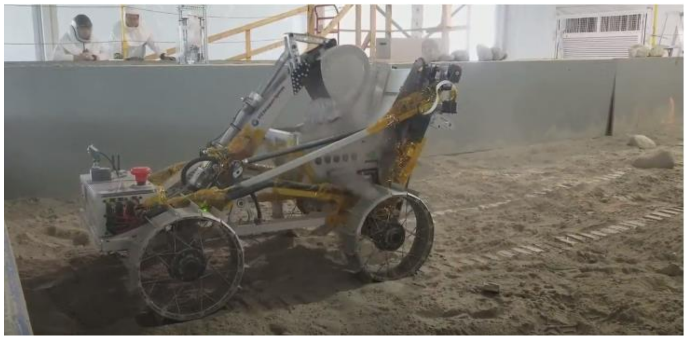


Figure 3: North Dakota State University's Robot

Coming in third place in the robotic mining competition is Kent State University. They produced a shovel design as their excavation system, digging regolith directly in front of the robot and carrying it to the loading station. This system required the robot to dig and return to the loading station multiple times instead of digging continuously for a long period of time. However, this system allowed enough regolith to be delivered to the point of Kent State University winning in third place.

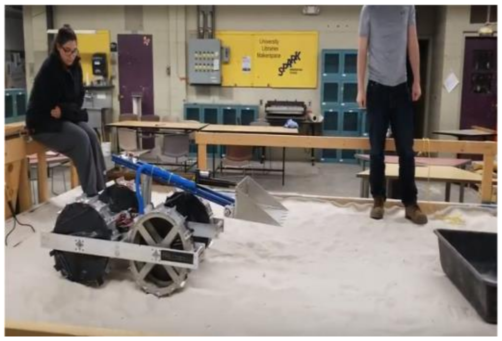


Figure 4: Kent State University's Robot During the Testing Phase

1.2.2 Current design

Ibex carries a similar design of Alabama's Astrobots, a bucket dredging system. The design was manufactured last year, as an improvement to the 2017 cycle under the robot of Markhor. The excavation system follows an effective design, but it was not implemented nor manufactured to a level of quality. This was the bottleneck of the robot during the competition. The focus of the problem stemmed from points, the design of the chain system and the construction of the bucket scoops themselves. The scoops were designed in a rushed fashion, as the rules changed halfway through the optimization of the robot. [3] The buckets were not fashioned to the chain, reducing the stiffness of the system, producing the problem of not digging into the regolith with enough force to dig efficiently.

The second problem of the chain system is also attributed to reducing the amount of force used by the scoops to dig into the regolith. There is a reduced amount of tension on the chains themselves, as well as low torque due to being powered by one motor. This was noticed during the competition, that the motor that operated the chain system drew too much power to complete its task.

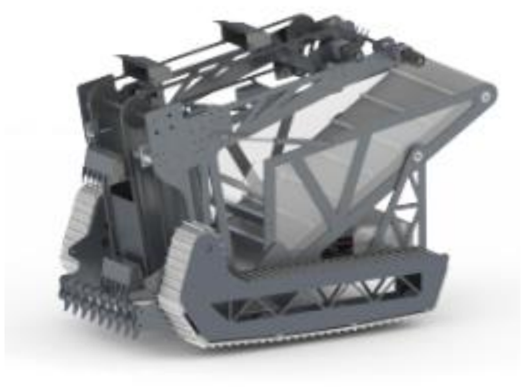


Figure 5:Ibex's CAD Design, 2017

1.3 Electrical Design

1.3.1 Current Design

On the current electrical system, the team from last year and two years ago used a fanless micro box PC running Linux, as well as a CTRE HERO motor controller and an Arduino UNO to interface with sensors and control servos.

The system is divided into four main subsystems: control station, robot software (the state machine), motor board and sensor board. The control station is the software on our PC to directly control the robot. The Robot software is run on a state machine on the robot that used Linux. Finally, the motor board is compiled of HERO motor controllers and a sensor board that utilized an Arduino. The HERO motor controllers' interfaces with a Talon SRX to send signals to motors, while the Arduino relays sensory data from the camera servos. The sensors are the IMU, limit switches, and bump switches. The messages from the sensors are sent to the control station via the robot software. The detailed diagram is included below:

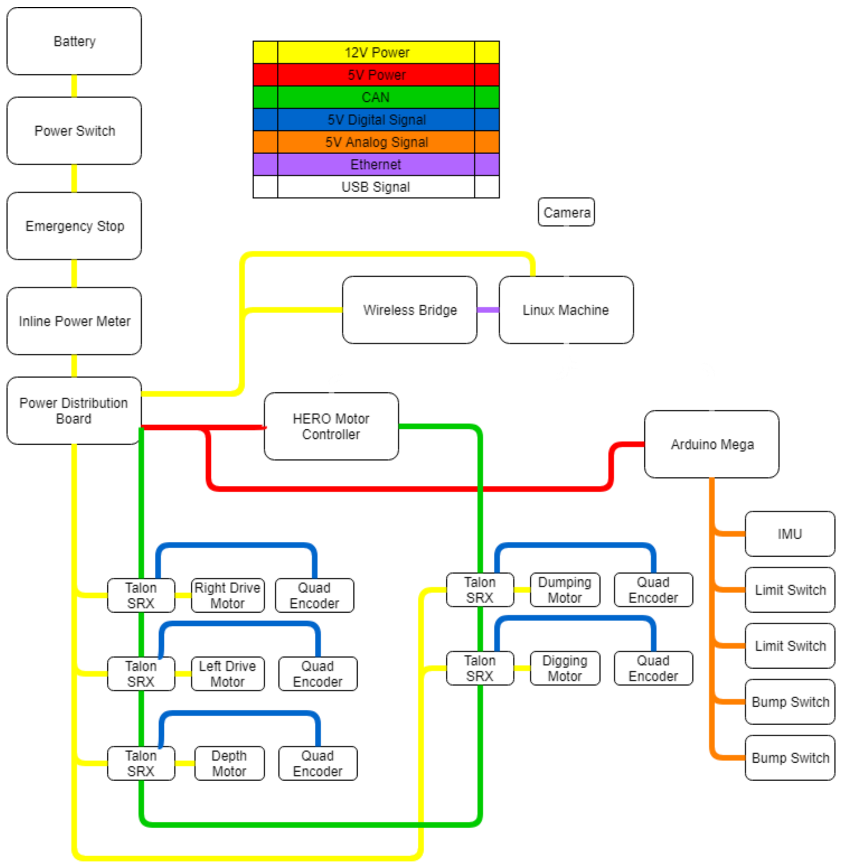


Figure 6: IBex Electrical Diagram

1.4 Software design

The robot is currently programmed using Python, C#, and Java. C# is used to program the HERO motor controllers, while JAVA is used for GUI and the rest of the robot is programmed in Python. An important factor of the design is the autonomy of the robot, as a fully autonomous robot is rewarded 500 points. To approach this challenge, we analyzed the pros and cons of each language, as well as frameworks that are commonly used for autonomy.

1.4.1 Python

Python is an easy language to implement into the robot, let alone control all the subsystems. In addition, Python has plenty of built-in libraries to interface with C/C++ and for other purposes [3]. That also helps with working across platforms such as Linux/Windows. In robotics design, it is common to have multiple languages in one robot. Python helps to resolve the issue of incompatibility between different systems. However, Python is not great when it comes to speed and multiple threading [4].

1.4.2 Robotic Operating System (ROS)

ROS is a framework for writing robot software and has been the most ubiquitous platform for many reasons. First, as we mentioned a robot has to interface with different systems, thus different software languages. ROS helps to mitigate this problem and make things communicate more efficient than regular APIs [5]. Second is modularity, ROS is structured to make every component modular since everything is connected by a distributed system. This means one system crashing doesn’t affect another system and messages are easily passed to different components. Finally, ROS has simulation tools such as Gazebo and Rviz where you can run the robot in a simulated world. In exchange for all the perks of using ROS, it has a deep learning curve and doesn’t have proper documentation in comparisons to other frameworks.

1.4.3 Current Design

The 2018 design has some automation programmed in Python. However, due to time limitation, last year’s team never got to test it out on the field. They had a queue of messages that would be sent over a socket connection from the control station (PC) to the Linux box on the robot. This acts as the state machine, where each state will have a corresponding logic that talks to the control station that the robot is ready to go to the next message in the queue. To communicate with the HERO board and the Arduino, they created separate threads to constantly talk over a serial connection to see if any of the incoming data was different from previous data. This data was used to determine in various places if the robot is ready to move on from the current message. To communicate with the control station, they had a TCP/IP socket connection open and talked back and forth with the control station that way.

To switch to ROS, there will need an overhaul of the current code but there are other advantages coming along with using ROS. Thus our next step was to evaluate whether our team was staying with Python or switching to ROS as a better choice.

Chapter 2: Methodology

For this year iteration, the name of the robot will be known as R.O.V.E. (Robotics Outer space Vehicle for Exploration). The steps required for this project to become successful what follows and R.O.V.E will be upgraded based on these objectives.

2.1 Analysis on IBex

To understand the issues as well as opportunities for development on IBex, our team first analyzed its performance mechanically and programmatically. Our initial analysis began with understanding how the robot operated without any form of external load. Initial testing was performed as the robot was manually controlled, while suspended on a wooden frame to allow the digger to operate without interference. At the current status under no load, the robot encountered issues relating to jamming and their sources. By initially separating individual problems causing the jamming to target, we were able to develop an understanding of how these issues contributed to the larger problem of digging inefficiency. During our analysis during testing, we found that the electronics and wiring were poorly performed. Without proper wiring instructions in addition to lack of electronics to operate the robot remotely, we operated the robot manually through powering specific motors directly. This did allow us to test individual systems directly for any mechanical failures without the dependence of any software related issues.

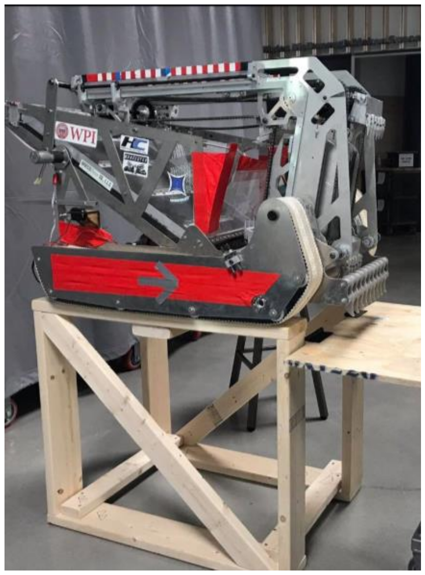


Figure 7: R.O.V.E. Propped Upon Testing Frame

2.2 Improvements for R.O.V.E.

After analyzing and evaluating ROVE's performance, we decided to focus on the following aspects.

2.2.1 Digging Mechanism

The main focus mechanically to improve are the bottlenecks of the excavation system, the chain system and scoops. Improving the fastening system of the buckets to the chain will reduce the sway of the scoops when digging the regolith. Currently, they are not fastened in an adequate fashion, which was the result of poor and rushed manufacturing processes.

A possible way to improve the stiffness is to increase the torque to the chain system, by adding a second motor or gearbox. Including a second motor would allow the excavator enough torque, and even to the point of having excess torque if needed, to dig without running into power draw related problems.

The intent of these improvements is to see the excavator dig down to the depth of 40 centimeters, reliably every attempt. In addition, this project wishes to measure the amount of regolith is excavated during the 10-minute duration.

2.2.2 Autonomy

During the competition, fully autonomous robot will be rewarded with 500 points. WPI's teams never performed any autonomy at the RMC due to other mechanical issues we faced. During 2018 competition, IBex did not run into any software problems. Therefore, we can move forward to implementing autonomy on the robot. As part of the competition, the robot should be able to cross the obstacle field, excavate, and return to the collection bin to deposit regolith. This is performed while the robot must be run autonomously for the full ten-minute round to receive full 500 points.

In order to achieve autonomy, we identified these main routines and possible feedback sensors that can be implemented.

Robot Self-Orienting

When the competition first starts, robot is facing a random direction. To self-orient itself, robot can use a LiDAR sensor to detect the BP-1 layer in the mining area and as well as the collection bin. Upon mapping out the location of the bin, robot would rotate itself to face the mining zone.

Crossing Obstacle Zone and Arriving at the Mining Zone

Once the robot figured out the its complete orientation, it has to drive across the obstacle zone. The robot's treads were specifically designed to drive over any obstacle, so there would be no need to drive around them. An IMU can be used to assist the robot to drive straight and achieve a desired distance. The length of the arena is known at the match, so we can input a distance prior to the start of the round. Once the robot reaches the set distance, it would then proceed to the next state.

Mining Regolith and Gravel

This step involves the robot to dig and collect as much gravel it can, up to its maximum capacity within the allowed time. This year, despite the mining field's formation of regolith and gravel, only gravel is rewarded with points. This means the 30 cm of BP-1 will be only excess material to dig through. Once the robot has reached the Mining Zone, the robot will extend the digging arm and drive the scoops. Upon extending up to 40 cm the robot will start a gradual drive in reverse, towards the collecting bin area while still collecting the gravel. An encoder will be used to know how deep the digger has extended and current draw will be taken into account in order to notify jamming.

Driving to the collection bin

As it collects and is driving backward to the collection bin area, the robot performs three checks simultaneously. If the hopper's capacity is reached, the robot has reached the obstacle area, or if enough time is not left to complete the run, the scoops will stop running and the extender retracts to its original position. The robot will then continue to drive backwards to the collection bin. The IMU is used to help the robot to drive back to the same location where it started, parallel to the bin.

Aligning to the Collection Bin

Once the robot is driven back to the collection bin, it would turn 90 degrees counterclockwise to be perpendicular to the bin. The robot would then start driving backwards again, until two bump switches are engaged against the bin. Bump switches are implemented to make sure the robot is in contact with the bin to avoid dumping imprecisely out of the bin. If one switch is not engaged, then dumping will not occur and the robot would adjust itself to align properly.

Dumping to Collection Bin

Finally, once both the bump switches are engaged, the robot will start running the conveyor belt to unload the contents from the bucket to the collection bin. The conveyor belt will run to an adequate speed for proper disposal of the contents. After the contents have been displaced, the robot will stop running the conveyor system. If there is time left even afterwards, the autonomous routine will restart.

2.2.3 A Dust Proof Design

As part of the competition grading criteria, designing dust proofing the robot's electronics and gearing system allow for additional points, up to 100 points for a completely dustproof and clean robot. The current design for dust proofing the electronics and gears involve plastic covers duct taped to enclose and air sealed. The problem arises when the robot needs to be worked on, and the electronics or gearing needs to have access to, the tape needs to be removed then replace. Replacing all old plastic covers with plastic tupperware would allow ease of access to any dustproof parts. Tupperware is waterproof, stopping the dust from breaching as well. In contrast to the plastic covers and duct tape, tupperware has a resealable cover that could allow quick access without destruction. As a secondary goal, dust proofing through this method would keep future improvements easier to perform.

Chapter 3: Ibex Detail Analysis

Through our initial analysis, we found here the shortcomings of the previous year's iteration of the robot.

3.1 System Analysis of Current Digging System

3.1.1 Gearbox selection

The first issue noticeable without testing was the combination between the selected gearbox and motor driving the digging system. The current motor utilized is a VEX 775Pro connected to a BaneBots P60 custom 256:1 planetary gearbox, which is not rated for safe operation with the 775pro [7,8]. During manual operation the robot underwent jamming, however the jamming issues always fixed itself as the motor attempted to stall then broke through the jam. After close inspection of the gearbox internals as well as the motor seen in figures 8 and 9, it appears that the motor at stall torque would slowly shear the internals of the gearbox. The input coupler as well as the first stage sun gear were damaged at their connectors. We found as the motor reached stall torque upon jamming, that the gears internally would slowly shear although only enough to allow the gearbox to still be functional.

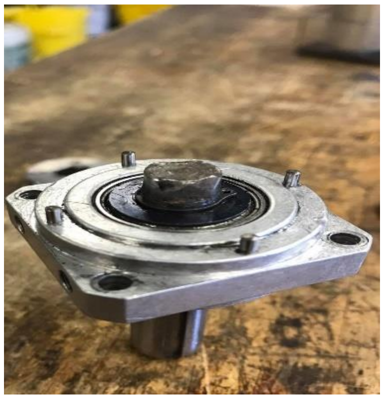


Figure 8: Output Shaft, BaneBot Gearbox

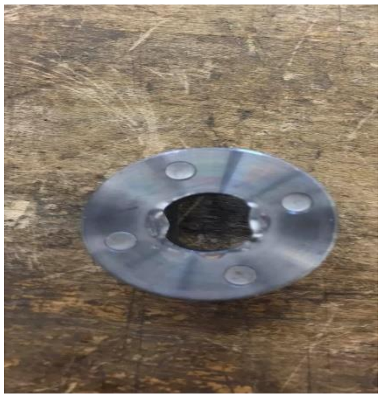


Figure 9: Connector Sun, BaneBot Gearbox

3.1.2 Fasteners for Scoops

Upon inspection of the scoops and how they were implemented, in addition to the reasoning behind their manufacturing allows some context as to the problems that these scoops had overcome while understanding the shortcomings of this design. The scoops were attached to the chain system four bar linkage design, using stock aluminum tabs connected to the scoops along with plastic spacers. Closer inspection of the tabs appear that the hole drilled for screws were not properly dimensioned nor manufactured in a standardized format. The tabs were manufactured poorly and in a rushed manner, using scrap aluminum without dimensioned cuts or slotted holes.

This design was originally created as to allow the scoops to traverse through the geometry of the chain drive system. Found in figure 10 can be seen a reflex point, which the scoops need to rotate over, then continue along its path to dump the gravel and BP-1. This was one of the points that the scoops were purposely designed to allow movement through the reflex point. In addition, there is a point near the end of the dumping path where the scoops travel through complex geometry, found in figure 11. Here is where the jamming issues had occurred, as previously mentioned. The design of the four-bar linkage did not take into consideration the placement of the sprockets and therefor causing jamming issues. The scoops jam as the geometry of the four-bar linkage is impossible to physically solve, however then it breaks from the motor reaching stall torque. The chain itself started to jump off the sprockets, both when the digging system was extended and retracted. From there, we tried to determine any possible variable changes. Using a camera with slow motion capabilities, we were able to record when the scoops and chain jumped and jammed. The scoops fasteners appear to have been damaged and we have identified this to be a large problem relating to the overall issue of the robot.

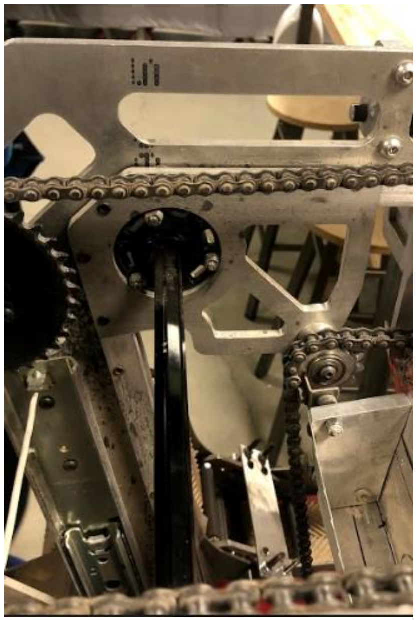


Figure 10: Reflex Curve

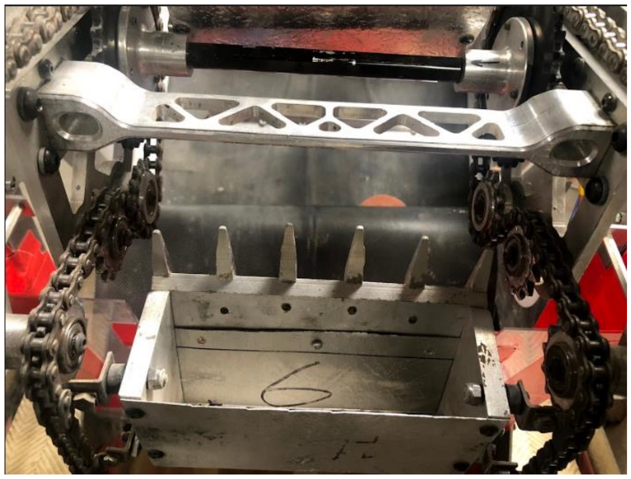


Figure 11: Complex Chain Path

3.1.3 Chain system flaws

The chain system that travels on the path for the scoops travels through complex geometry of the robot, shown previously. However, the chain does not suffer any problems at this point. The chain becomes damaged, however, through machine working on the opposite end of the robot. The chain itself, when undergoing the digging process, machine works against the side plate frame of the digging system which would increase drag forces along the path of the chain, additionally increasing the tension on the tape drive of the digging system. This forces the motor to increase the amount of power and torque in the system.

3.2 Control System Analysis

Our first attempt was to develop autonomy based on the existing control system. However, as we started to analyze deeper into the existing control system, we begun understand how complicated the . The next question is whether we want to try to understand the complexity and develop on it or start from scratch.

3.2.1 Hardware Complexity

Before understanding the code, we had to first understand the structure of the control system. As mentioned above, the previous control system was extremely complex. On the robot alone there were a Linux box, a HERO Board, and an Arduino just to control the functionalities of the robot. This created a problem when testing as every piece of hardware has to be powered through different sources, while communication ports have to be opened and ready to communicate. To mitigate the issue, developed on the reasons as to the function of each piece of hardware and if we could consolidate under one system. We wanted to answer two questions:

1. Can the Talon SRX be connected directly to the Arduino so we can remove the HERO Board?
2. Can the Arduino communicate with our control station so we can remove the Linux box?

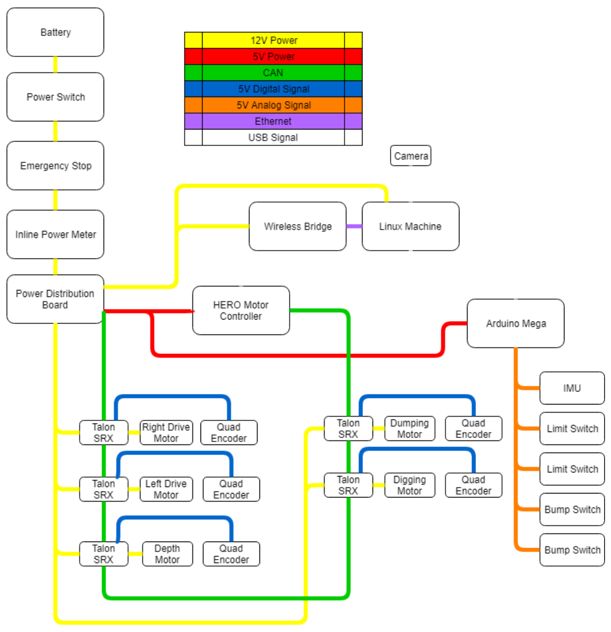


Figure 12: IBex Electrical Diagram

As an answer to the first question, Talon SRX can be wired to Arduino board but indirectly through an Arduino Shield, another board. Thus this would not be an option. Arduino can communicate to a PC through serial port to replace Linux Box. However, Arduino's performance is slow and will be inefficient to run the autonomous program. Additionally, switching to Arduino will requires a rewrite of the entire robot's machine that is currently in the Linux Machine in the Arduino environment.

The second setback we encountered when we took over the robot was the fact that all the electronics were unwired by the team before us. After 2018 competition, all the wires got terribly tangled, seen in figure 14, so the team decided to take apart the hardware with an attempt to rewire them but unfortunately, they did not. The schematic was not detailed enough for us to identify where the wires should be connected to. Thus, this motivated us to produce a dust-proof and more secured housing for our hardware.

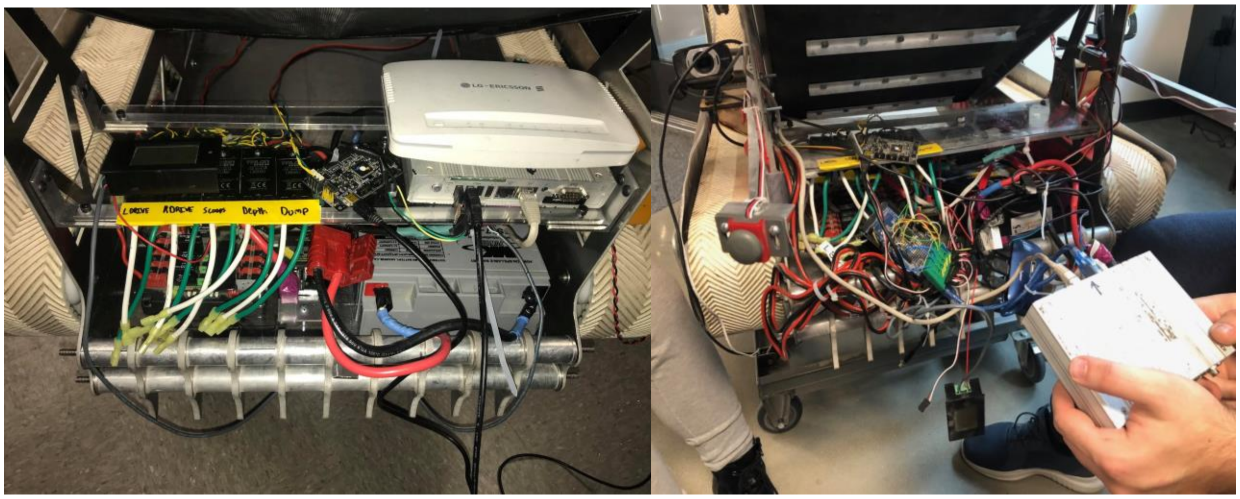


Figure 13: IBex Control System Before and After 2018 Competition

3.2.2 Software Complexity

We were able to ask for the Github repository of IBex. The code itself was neat. However, there was no instruction on how to power the control system up or how wires are connected. Additionally, we could not understand how programs communicate with one another, programmatically. Much effort was dedicated to reading and understanding the code and how exactly messages are being sent from Control Station to Motor Board and vice versa.

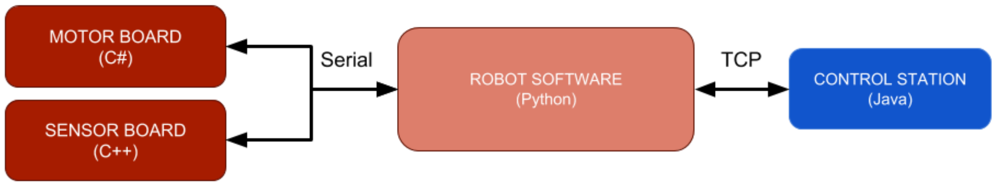


Figure 14: IBex Software System and Communication

Chapter 4: R.O.V.E.'s Mechanical Design

4.1 Mechanical Design Considerations

4.1.1 Excavator feature requirements

Our team decided that ROVE would be best to modify the current design for the scope of this project instead of creating an overhaul redesign of the system. After locating the features need to be considered for our current design and how it would be implemented into specifications involving the rules of the competition. We identified key features to be addressed shown in table 1.

Table 1: Features and Their Shortcomings

Scoop	Motor
Fasteners connected to scoops and chain	Improper gearbox
Lack of support for cantilever forces	Prevent potential jamming to reduce damage

Our modification would require that the digging depth be 40 cm from the surface of the ground, enable to reach the gravel located after 30 cm of initial depth. The scoops need to be supported during the digging process, which produces heavy cantilever forces on the chain. This would require either more fastening points of the scoops to the chain to reduce the load at each point or to develop a support for the scoop to prevent deflection during engagement with the ground. This needs to be performed while still allowing the scoops to rotate and travel around the driving sprockets and the complex chain path.

4.2 Design Modifications

4.2.1 New Gearbox Selection

The original gearbox was not rated for a 775Pro motor, which had initial problems with the safety rating for the motor. Therefore our team had purchased a gearbox designed for the 775pro, the Versaplanetary gearbox. Our ratio for the motor was 250:1, compared to the 256:1 BaneBot gearbox. We selected this gearing ratio based around the motor we were using, in addition to the safety rating of the gearbox itself at the selected ratio. This is to prevent any damage the motor would cause to the gearbox in case of potential jamming, similar to last year's scenario. To accompany the new motor, we redesigned the mounting plates as well as replaced the customized sprocket hubs with standardized hubs for ease of implementation.

4.2.2 Fastener Design

After understanding the initial conditions of why the previous year's fasteners were designed, we decided to change the design of the fasteners from a rotating linkage to a singular tab, inlaid in with scoops themselves. This would allow our team to simulate the scoops as a singular cantilever beam, as the scoops become a singular rigid body with the chain. Seen in figure 16, we produced these tabs from plain carbon steel, as the reactionary forces applied to the tab during digging engagement would cause aluminum tabs to fault.

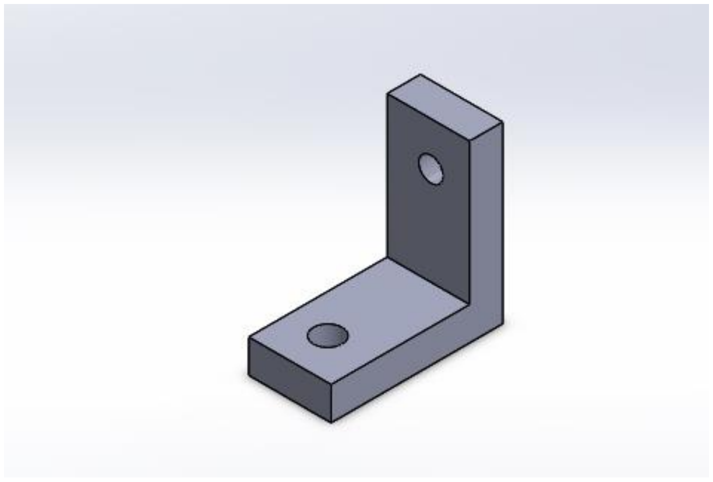


Figure 15: Fastener Tab Design, Solidworks

4.2.3 Delrin Chain Guide

To provide the support to the scoops, needed when engaging into the ground, our team manufactured a custom closed faced chain guide utilizing Delrin as the material. This profile was designed to encompass the entirety of the roller chain excluding only the tabbed face. We selected Delrin for its self-lubricating capabilities, in addition to the ease of machining. We utilize the softness of Delrin as our form of tolerancing while digging. If potential regolith or BP-1 attempts to enter the chain guide, then as the chain travels through the guide the regolith or sand would get pushed along and potentially machine through the profile. This would allow even smoother use as the profile expands but also helps clean the chain guide to reduce jamming. The custom profile was designed solely to encapsulate the master links attached to the chain, as chainguides on the market have a profile design for only simple roller chain. The profile was created to follow the original path of the chain when the sprockets were originally attached, which can be found in figure 17.

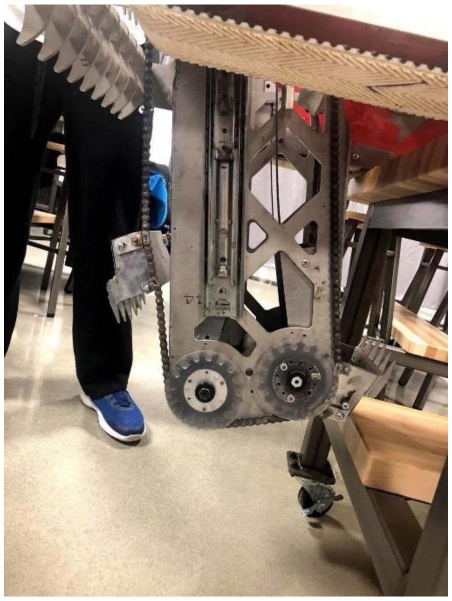


Figure 16: Ibex Digging Engagement Sprockets

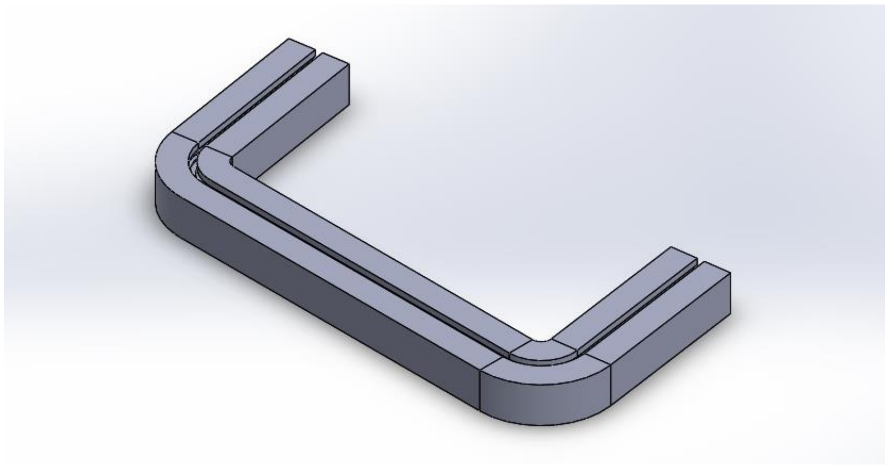


Figure 17: Delrin Chain Guide, SolidWorks

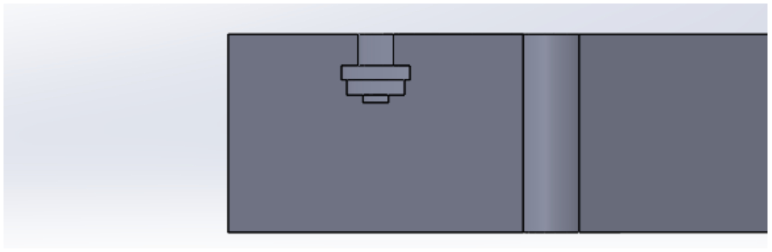


Figure 18: Chain Guide Profile, SolidWorks

A new excavator side plate was designed for the purpose of holding the chain guides in place. The design was formed from the previously used pattern, originally created to help reduce weight. Figure (3) shows the final product with the chain guide attached.

4.2.4 Mechanical Analysis

After determining the shape of the scoop system, our team simulated a simple rigid body design in SolidWorks to determine the stress analysis and forces involved while digging. We calculated the force from the motor applied to the chain using force statistics and created a basic analysis. We determined the motor to be running during the competition at 16A, using data retrieved from previous year's competition. The motor would have an output torque of 0.72 in*lbs., in which we could determine the forces applied to both the scoop, tabs and chain.

If: 18.5 A = 0.84 in.*lbs. 9.6 A = 0.42 in.*lbs.

=> 0.72 in.*lbs. = 16 A

Torque ratio: 250

Tau = 250 (0.72)(0.9) = 162 in*lbs

36 teeth ⅜ radius

Force applied to distance = (36 * ⅜) / (2 * pi) = 2.15 in

Fc = 75.35 lbs applied to the scoop

We theorize the force applied to the scoop will be approximately 75 lbf maximum to the tip of the teeth. Using SolidWorks simulation, we created a singular simple mesh of the scoops, tabs and chain. We then fixtured the chain ends in proper orientation to how it would interact within the chain guide. The only points of possible failure found in this simulation were on the chain tabs themselves, on the corner connection to the tab from the chain. However, beyond this potential mode of failure, the rest of the simulation represented that the scoop and new tab would not deform under these predicted forces.

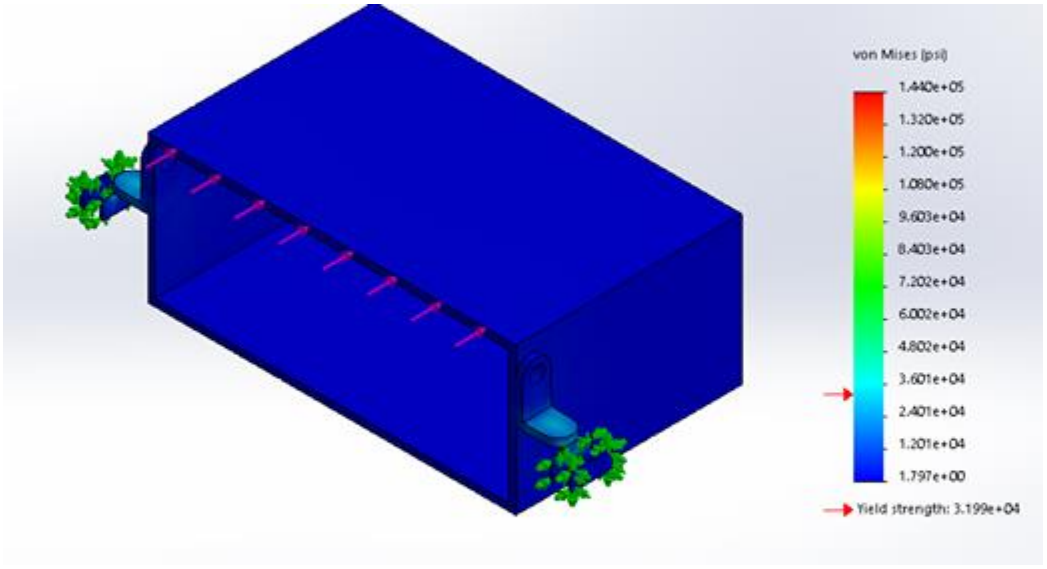


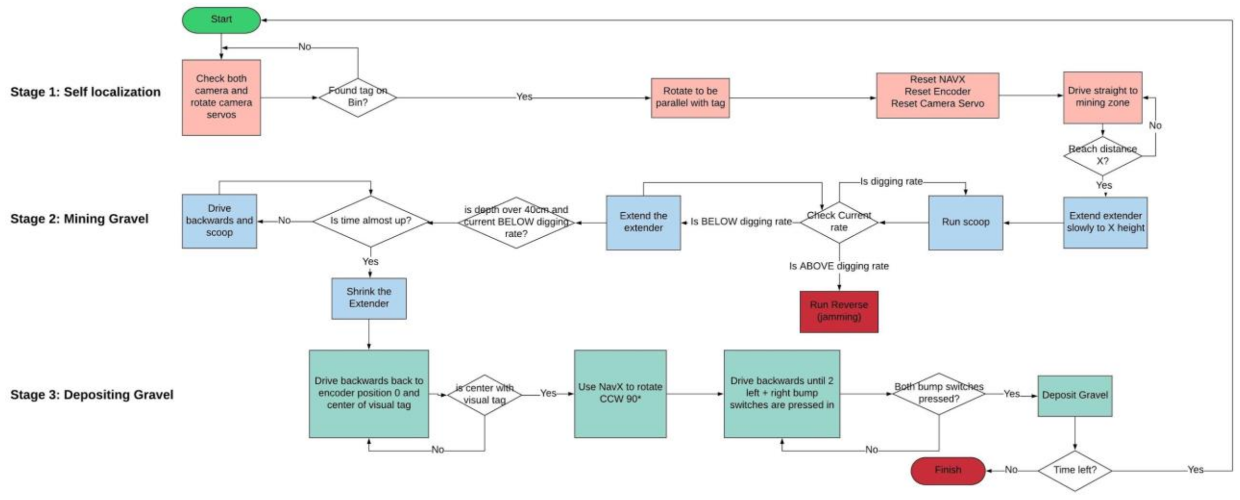
Figure 19: SolidWorks Simulation of Forces Applied

Chapter 5: R.O.V.E.'s Autonomous Design

A major focus for this year's robot programmatically is the implementation of autonomy. After being able to understand and analyze the previous program, we could see that it was partially autonomous where computer vision could identify the if the robot is offset the left or the right of the goal image to align perfectly to the goal. A set of tasks can be queued up to execute. However, there were no feedback sensors implemented for automated activities to run timely in the right behaviors. With all the issues we identified above, we decided to completely swap out last year's control system to make autonomy more accessible.

5.1 Autonomous Operation

The attached flow chart describes the autonomous operations in detail.



Start
Stage 1: Self localization
Check both camera and rotate camera servos
Found tag on Bin?
No
Yes
Rotate to be parallel with tag
Reset NAVX
Reset Encoder
Reset Camera Servo
Drive straight to mining zone
Reach distance X?
No
Yes
Stage 2: Mining Gravel
Extend extender slowly to X height
Run scoop
Check Current rate
Is digging rate
Is BELOW digging rate
Is ABOVE digging rate
Run Reverse (jamming)
Extend the extender
is depth over 40cm and current BELOW digging rate?
Is time almost up?
No
Drive backwards and scoop
Yes
Shrink the Extender
Stage 3: Depositing Gravel
Drive backwards back to encoder position 0 and center of visual tag
is center with visual tag
No
Yes
Use NavX to rotate CCW 90*
Drive backwards until 2 left + right bump switches are pressed in
Both bump switches pressed?
No
Yes
Deposit Gravel
Time left?
No
Yes
Finish

Figure 20: R.O.V.E's Autonomous Routine Flowchart

The autonomous operation is simplified into three main focuses.

5.1.1 Self-Orientation

At the start of the competition, the robot will be facing a random direction. Thus, the first step of the autonomous operation is to have the robot identify what direction it is facing and therefore rotate to aim towards the mining zone. The self-orientation process is assisted by two cameras placed on left and right of the robot, each camera is mounted on a servo, and a visual tag attached in the middle of the collecting trough. The collecting trough is on a side of starting zone.

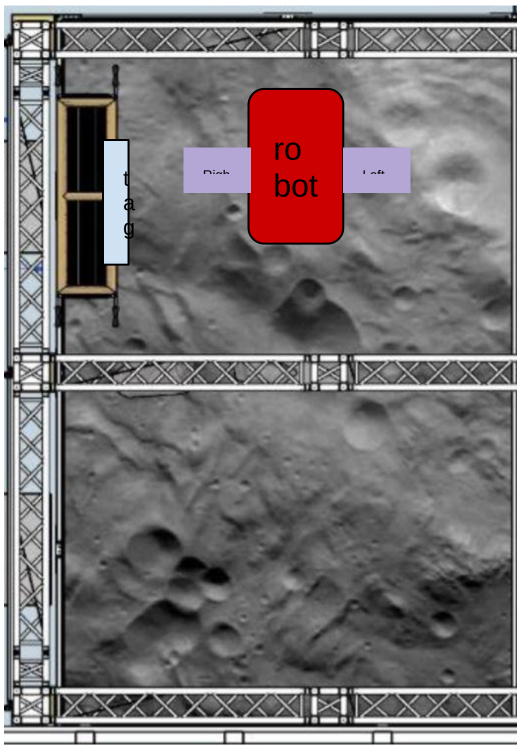


Figure 21: One Half of RMC Arena, NASA 2019

There are six possible positions that the robot will be facing:

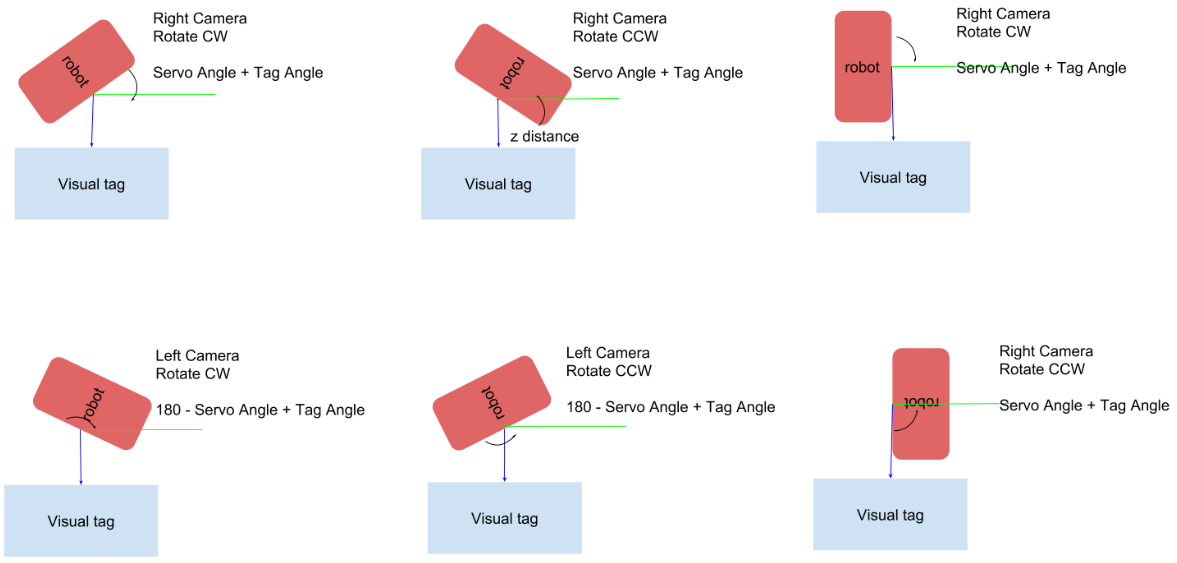


Figure 22: Possible Starting Orientations of the Robot

Depending on which side of the robot the trough will be placed, that side's camera will run first. Right now, we are assuming the trough will be on the right of the robot for this year's competition. Thus, right camera will run first. Each camera will scan at 0, 45, 90, 135 and 180 degrees servo angle to make sure the tag is identified. If the right camera does not capture anything, the program will run the left camera and know that the robot not facing forward so the robot will have to turn a larger angle to face the right direction - meaning it faces the mining zone.

With the visual tag that we use, the robot can align itself parallel to the trough. Once it is aligned, the robot will reset its navX, encoder, camera servos and get ready to drive straight to the mining zone. An encoder is mounted on the drivetrain motor shaft to assist drive distance.

Notice that here we discard driving through the obstacle zone since the robot was designed with treads to drive over the obstacles and the cow catchers push the obstacles out of the way.

5.1.2 Mining Gravel

Mining logics have to take into consideration how deep the extender has extended and how much current the scoop is drawing while digging. It is important to run multiple manual-operated tests prior to autonomy to note the robot’s digging current on regolith and gravel. For example, from Markhor, we learn that when the scoop is not digging (not fully emerged in sand or gravel), it drew around 1.5A. When the scoop dug only sand it drew on average 6.5A and gravel from 17-18A. Knowing whether the scoop is digging or not is important so we can extend the extender down more. In the competition arena, gravels will be found from within 30 cm of depth and gravel is found from 30 to 40 cm. Once the extended has reached 40 cm (maximum depth) and the scoop signals that it is not digging anything anymore, that means we have dug everything available at the position, we will start slowly driving backward to dig more gravel at the depth of 40 cm. We won’t be driving to different spot, to avoid going through the process of digging through regolith again since only gravel is rewarded with points.

One of the techniques we learned to avoid jamming programmatically is to run the motor reversely. When the scoop motor controller - Talon SRX - senses that the current is going pass the gravel digging range and reaching the jamming level. The program will automatically detect the signal and will run the motor reversely for two seconds.

Similar to drive distance, an encoder is mounted on the extender’s shaft to convert ticks to meters to monitor how deep the extender has extended.

5.1.3 Depositing Gravel

Timing is taken into account during the whole process. The robot will dig until time is almost up but still enough time left to drive back and deposit gravel.

The robot will drive backwards to where it starts - where the encoder position is 0. Once it is back to the starting zone, it will turn 90 degrees CCW and start backing until both bump switches are pressed in. Once both bump switches are pressed, it will start depositing the gravel.

5.2 Electrical Components

When we were trying to research a way to simplify the robot control system, we found that besides the HERO board, a roboRIO board can also connect with Talon SRX through CAN bus. In addition, there are other pros that will be discussed in this section. Therefore, we decided to overhaul the entire robot control system and start from scratch.

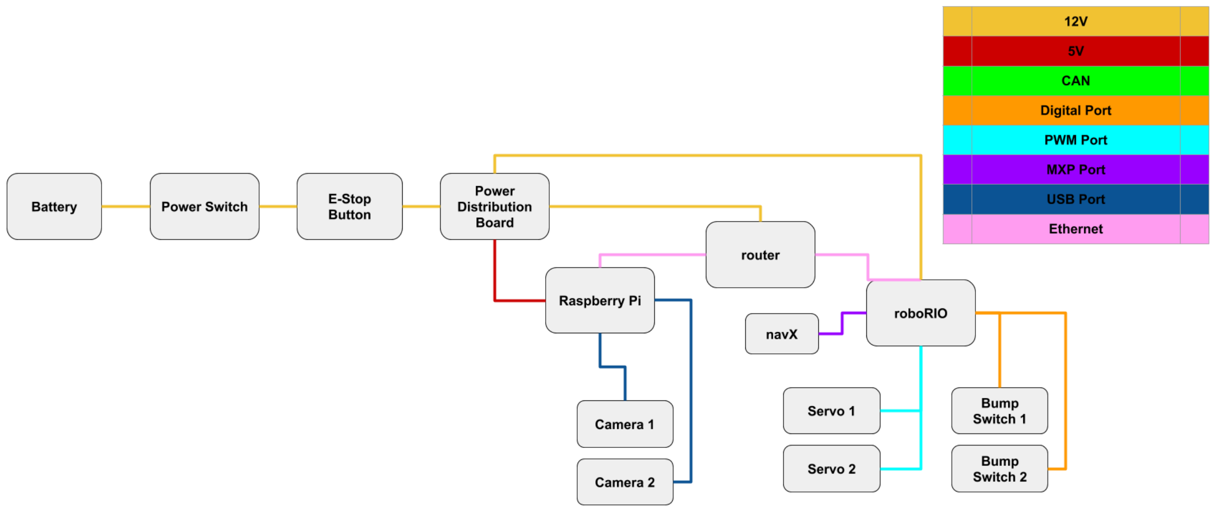


Figure 23: R.O.V.E Electrical System

5.1.1 Main Computer - RoboRIO

RoboRIO, made by National Instruments (NI), is a robot computer that can interface directly with our computer and the motor controllers Talon SRXs. In addition, it also has built-in ports for cameras and sensors like switches and the IMU. The roboRIO will replace the previous Linux machine, HERO board and Arduino. RoboRIO board allows faster integration and simpler programming system. We will only need one control system rather than four different programs like before.

RoboRIO supports Java, C++ and Python. It has its own built in libraries to support the integration with motor controllers and sensors.

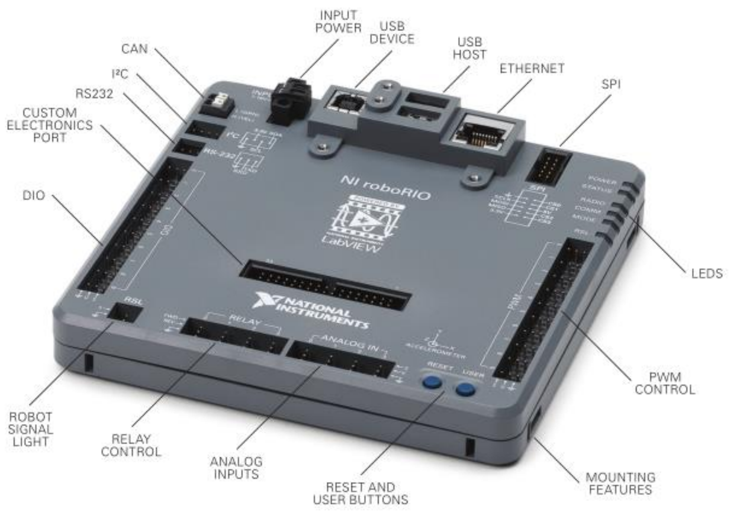


Figure 24: roboRio, National Instruments 2019

5.1.2 Localization - Raspberry Pi and Cameras

Choosing the right computer vision system is essential to complete the localization process. We identified options for sensors for localization. We originally planned to use a

LiDAR to map out the field. However, after our research, we realized LiDAR is more costly and provides higher functionalities than we need. Thus, we decided that using a camera vision is adequate for our needs. We reused the cameras we had from last year, the Logitech HD 1080p cameras.

The cameras will be placed on the left and right side of the robot to completely scan 360 degrees of the robot's surrounding. The roboRIO has USB ports for cameras. We were originally running the vision system on the roboRIO. However, after identifying the visual tag that we will use to support the robot's self-orientation - ArUco marker (this is covered in section 5.3 Computer Program), developed by OpenCV's extended libraries - Opencv-Contrib, we ran into an issue where the roboRIO does not support the installation of opencv-contrib library. To solve the problem, we use a Raspberry Pi 3 B+ as a co-processor to run the vision system. Using Raspberry Pi to run the computer vision gives us a advantage of reducing CPU usage on roboRIO so the RIO can focus on other autonomous operations and running a faster vision system on Raspberry Pi.

Figure 25: Raspberry Pi 3 B+, Raspberry Pi

To physically communicate, the Raspberry Pi and the roboRIO both are plugged into an ethernet cable to the same router. We gave both the Pi and the RIO a static IP address. Data is sent to a NetworkTables which is discussed in section 5.3.3 Communication.

5.1.3 Feedback sensors - IMU, Encoders and Servos

5.1.3.1 NavX as IMU

Autonomous operations require the robot to report its heading and the digger's depth. To track the robot's angle and driven distance, we will use a navX-MXP, which is a 9-axis inertial/magnetic sensor and motion processor. One of its features is measuring robot's roll, pitch, yaw angle. Moreover, the navX integrates directly with the roboRIO. It can be plugged into MXP port on top of the roboRIO without any wire.

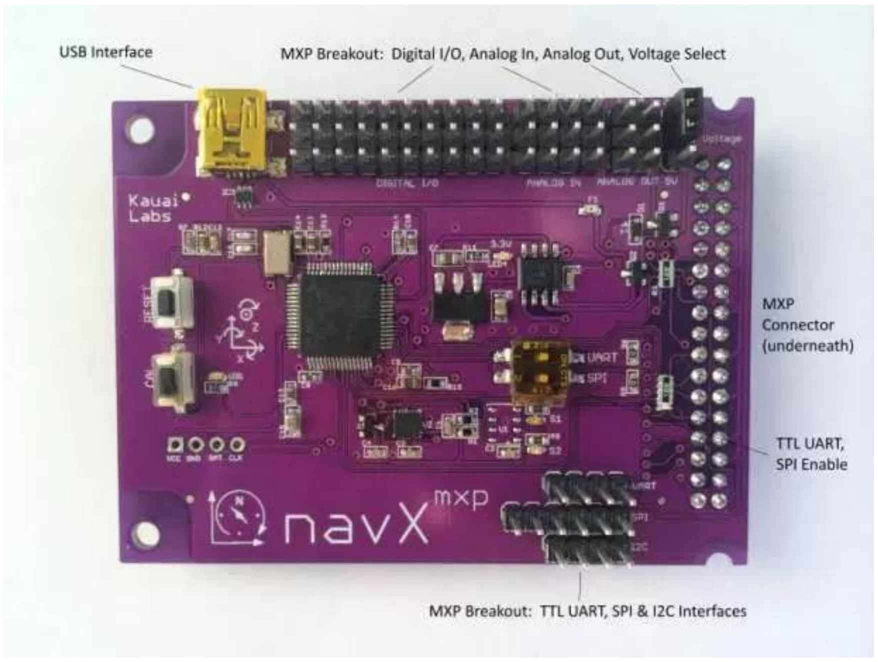

USB Interface
MXP Breakout: Digital I/O, Analog In, Analog Out, Voltage Select
MXP Connector (underneath)
TTL UART, SPI Enable
MXP Breakout: TTL UART, SPI & I2C Interfaces
Kauai Labs
navX mxp

Figure 26: navX-MXP, Kauai Labs

5.1.3.2 Encoder SRX

In order to increase accuracy for robot's heading and also how far the digger has extended, we use encoders to mount on the motor's shaft on one end and wire with Talon SRX motor controller on the other end. The quadrature SRX encoders are produced by the same company that makes the Talon SRX so they are compatible and easily physically and programmatically connected.

Figure 27: SRX encoders, VEX Robotics 2019

5.1.3.3 Servos

Cameras are positioned in a way such that each of them can scan 180 degrees on the left and on the right of the robot. Unfortunately, our cameras' maximum field of view is only 76 degrees. Therefore it won't be seeing a complete 180 degrees. Instead of rotating the robot around until the cameras catch the visual tag, we will place the camera on servos to reduce power usage from running the drive motors. The servos that we use are the Hitec HS-311 servos that can turn 360 degrees. The servos can be wired directly onto the roboRIO through PWM ports.

Figure 28: Hitec HS-311 servo, 2019

5.3 Computer Program

Despite our original plan of using Robotics Operating System (ROS) as our main, we decided to move on with another option due to ROS complexity for first-time robotics programmer. We implemented MagicBot for the robot program [11]. MagicBot is a Python framework for roboRIO environment, designated for creating robot programs and often used in FIRST Robotics Competition. roboRIO supports Java, C++ and Python. However, the newly developed Python libraries have a simple simulation where robot code can be run and tested.

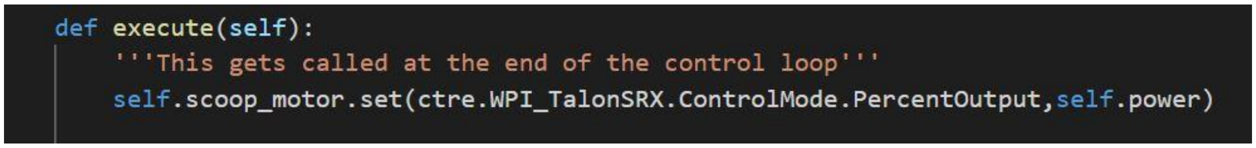
5.3.1 MagicBot Framework - Low Level

In this program, we created six components to control six main parts of the robot: Drivetrain, Extender, Scoop, Dump, Camera and Camera Servo. Detailed UML is attached in Appendix A. Each component consists of different functions, but there are three main types of functions:

- Control methods
- Informational methods
- An `execute` method

`execute` Method

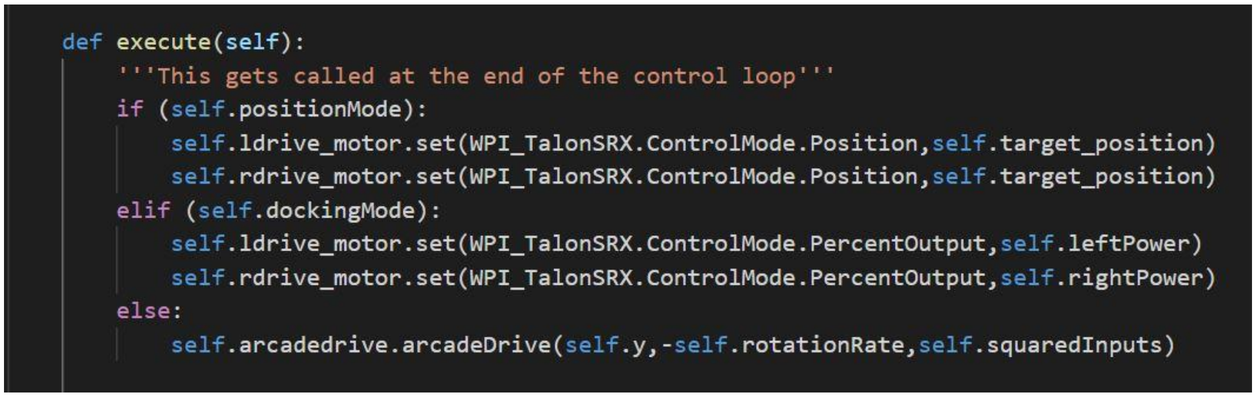
In MagicBot framework, `execute` method is called periodically by the machine to send data directly to the device, such as setting the power to the motors. It is the lowest level of programming where function interacts directly with the devices. In our program, `execute` method looks like this



```
def execute(self):
    '''This gets called at the end of the control loop'''
    self.scoop_motor.set(ctre.WPI_TalonSRX.ControlMode.PercentOutput,self.power)
```

Figure 29: Code Snippet of an Execute Method in Scoop Module

Or more complicated:

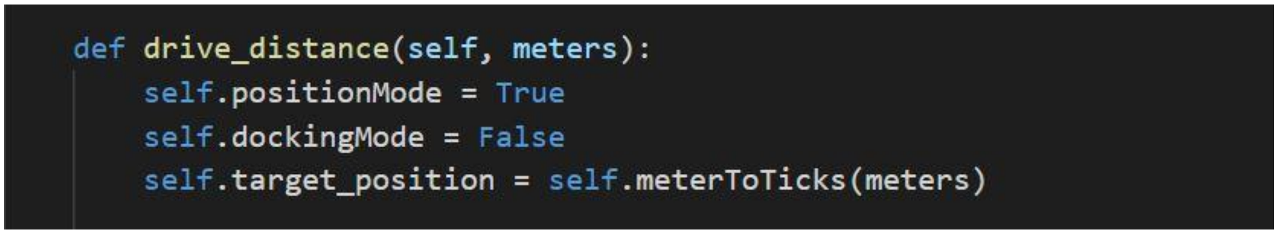


```python
def execute(self):
    '''This gets called at the end of the control loop'''
    if (self.positionMode):
        self.ldrive_motor.set(WPI_TalonSRX.ControlMode.Position,self.target_position)
        self.rdrive_motor.set(WPI_TalonSRX.ControlMode.Position,self.target_position)
    elif (self.dockingMode):
        self.ldrive_motor.set(WPI_TalonSRX.ControlMode.PercentOutput,self.leftPower)
        self.rdrive_motor.set(WPI_TalonSRX.ControlMode.PercentOutput,self.rightPower)
    else:
        self.arcadedrive.arcadeDrive(self.y,-self.rotationRate,self.squaredInputs)
```

Figure 30: Code Snippet of an Execute Method in Drive Module

Control methods

Control method are low level "*verb*" functions that provide useful abstraction such as drive_forward, stop, rotate. These control methods store important values to perform a desired action. However, this don't directly execute the action, they only set the values such as power and then `execute` method will carry the action out. Control methods are called by and receive input data from the high level:



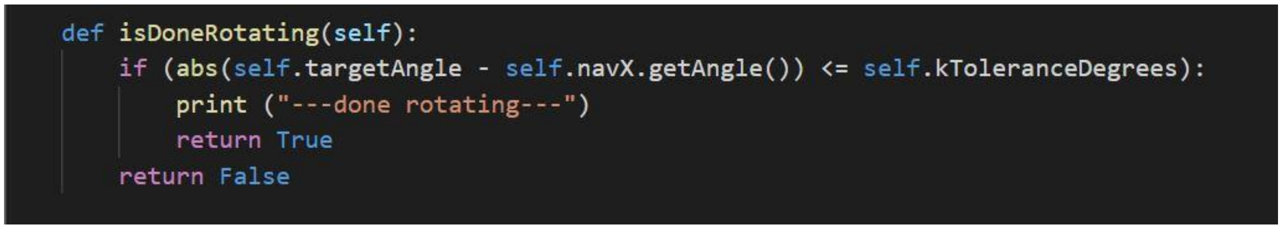
```python
def drive_distance(self, meters):
    self.positionMode = True
    self.dockingMode = False
    self.target_position = self.meterToTicks(meters)
```

Figure 31: Code Snippet of a Control Method in Drive Module

Here target_position is calculated from meters input from higher level and then get sent to execute method to signal the motor controllers to drive a certain distance.

Informational methods

These are methods that are used to tell something about the components, typically used to check if previous activity was done or if the condition is satisfied for next activity to be carried out. For us, we have these to check whether the robot has rotated a given angle or whether goal is found before driving forward and so on.

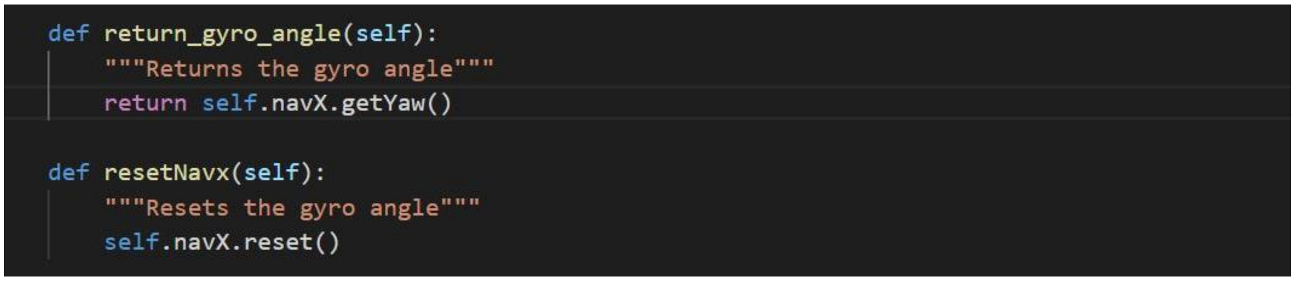


```
def isDoneRotating(self):
    if (abs(self.targetAngle - self.navX.getAngle()) <= self.kToleranceDegrees):
        print ("---done rotating---")
        return True
    return False
```

Figure 32: Code Snippet of an Informational Method in Drive Module

Figure 31 is showing an informational method in Drive module to check if the robot is done rotating and came to complete stop before moving onto the next state to avoid driving while still rotating.

We also have informational methods to solely return a sensor's value or reset a sensor so that the high-level modules won't directly interact with devices when they need to check values

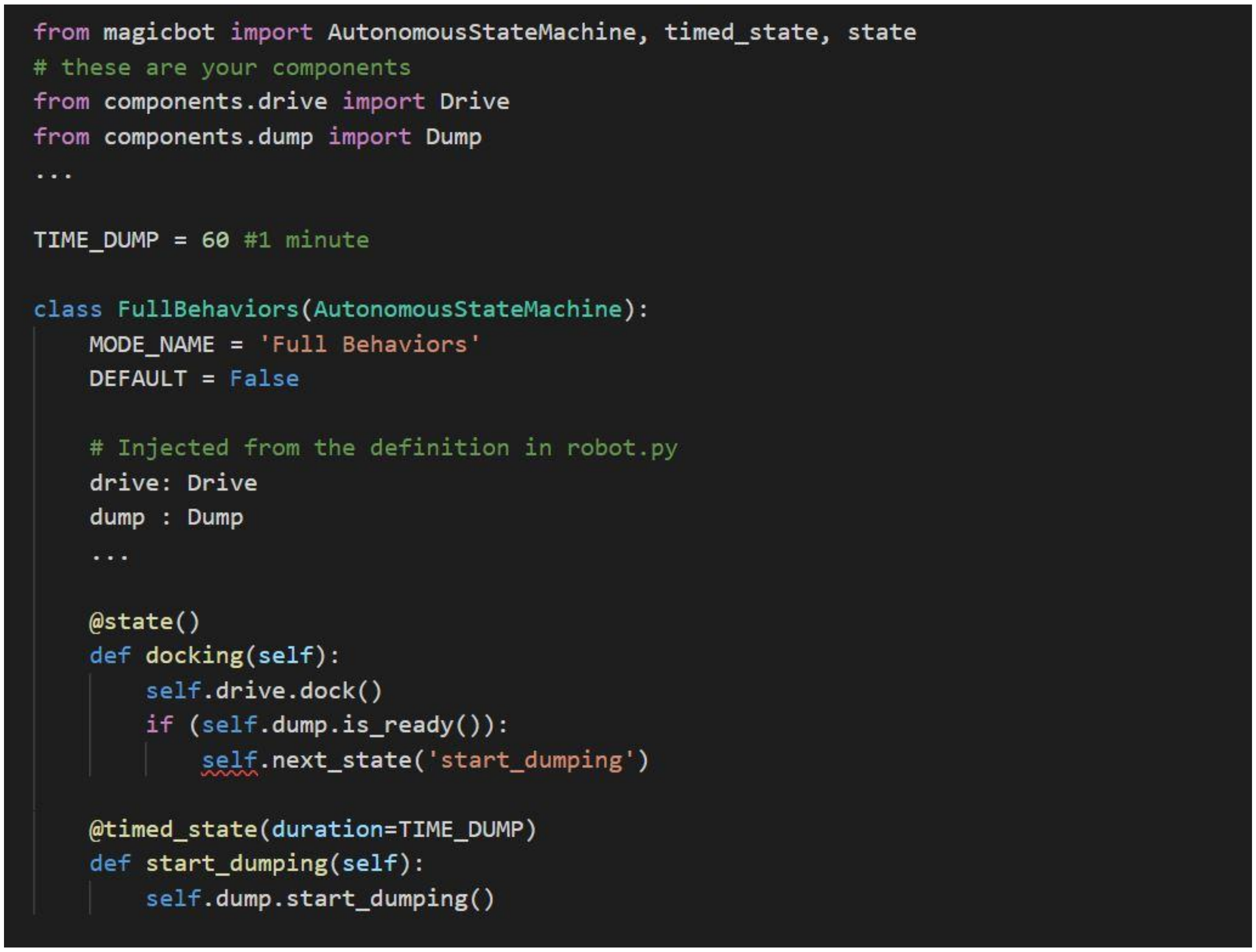


```
def return_gyro_angle(self):
    """Returns the gyro angle"""
    return self.navX.getYaw()

def resetNavx(self):
    """Resets the gyro angle"""
    self.navX.reset()
```

Figure 33: Code Snippet of other Informational Methods in Drive Module

5.3.2 MagicBot Framework - High Level

High level components are those that control other components to carry out the automated behaviors. This can be view as the state machines. Each component can be set to run a certain time as timed_state or a state run until informational method returns a satisfactory condition to switch to a next state



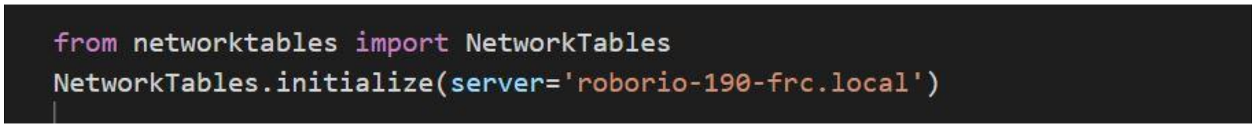
```python
from magicbot import AutonomousStateMachine, timed_state, state
# these are your components
from components.drive import Drive
from components.dump import Dump
...

TIME_DUMP = 60 #1 minute

class FullBehaviors(AutonomousStateMachine):
    MODE_NAME = 'Full Behaviors'
    DEFAULT = False

    # Injected from the definition in robot.py
    drive: Drive
    dump : Dump
    ...

    @state()
    def docking(self):
        self.drive.dock()
        if (self.dump.is_ready()):
            self.next_state('start_dumping')

    @timed_state(duration=TIME_DUMP)
    def start_dumping(self):
        self.dump.start_dumping()
```

Figure 34: Code Snippet of a State Machine Module

Here is a code snippet from our main state machine. We import the low-level components that will be used such as Drive and Dump. There are two states: docking and start_dumping. However, before transitioning to start_dumping state, our state machine has to check whether it is ready to dump (whether the robot is aligned perpendicular and touching the collecting trough) before calling the Dump component to execute the dump motor.

5.3.3 Communication

To communicate across components and to external program such as our vision system on Raspberry Pi, we use NetworkTables. This can be thought of as a common table that is shared among processes. This table contains values that will be updated continuously. Values can be get or set by any components. NetworkTables can be connected through static IP address 10.XX.XX.2 or mDNS Hostnames roborio-XXXX-frc.local where XXXX is FIRST team number. Ours is 190



```
from networktables import NetworkTables
NetworkTables.initialize(server='roborio-190-frc.local')
```

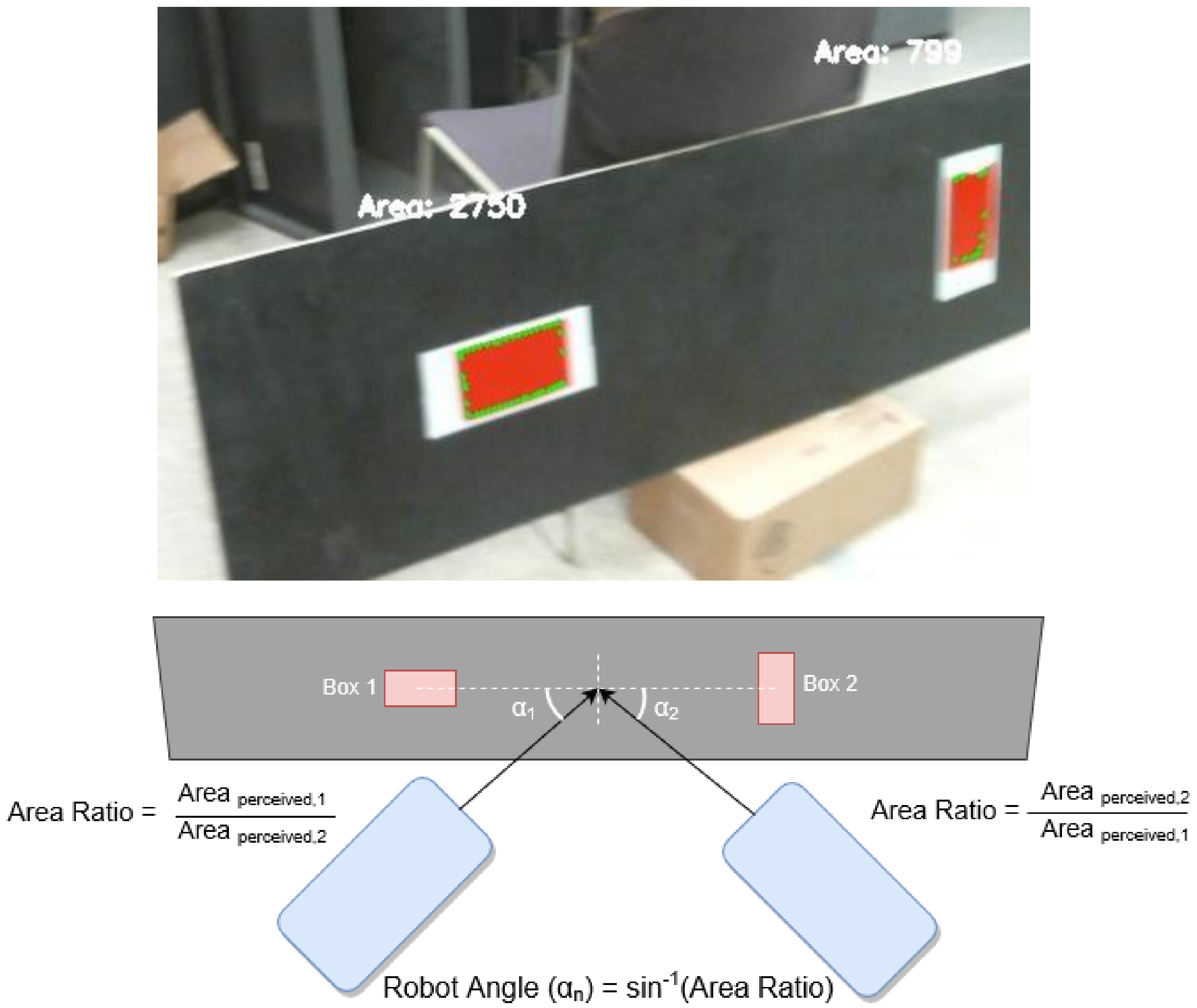
Figure 35: Code Snippet of Server Connection

In our program, we update to NetworkTables the devices' values such as the Talon SRX's current drawn or the navX's yaw angle to use by other components.

SmartDashboard is a software by NI installed our computer to interface with roboRIO. Data that is sent via NetworkTables is viewed on SmartDashboard.

5.3.4 Computer Vision

Computer vision was one of the main focuses and an essential step for the robot to operate robustly during self-localization process. In this process, we will be using OpenCV library to process images. We also acknowledged that 2018 team developed a vision system that used two-shape image as a target to identify which side (left or right) the robot is offset and thus calculate the angle.



Area: 799
Area: 2750
Box 1
Box 2
α_1
α_2
Area Ratio = $\frac{\text{Area}_{\text{perceived},1}}{\text{Area}_{\text{perceived},2}}$
Area Ratio = $\frac{\text{Area}_{\text{perceived},2}}{\text{Area}_{\text{perceived},1}}$
Robot Angle (α_n) = $\sin^{-1}$(Area Ratio)

Figure 36: 2018's Computer Vision

This year, our goal was to choose a target image that can be easily recognized under dusty condition and also can report the angle and distance from camera to image. We could have developed from last year's work, however after researching we found a visual tag called ArUco tag that has built-in functions to support pose estimation. ArUco is a fiducial marker that provides fast and robust detection with a single tag. ArUco library can be implemented from opencv-contrib library (extended library of the regular Opencv) - this was also the reason why we couldn't run ArUco on roboRIO.

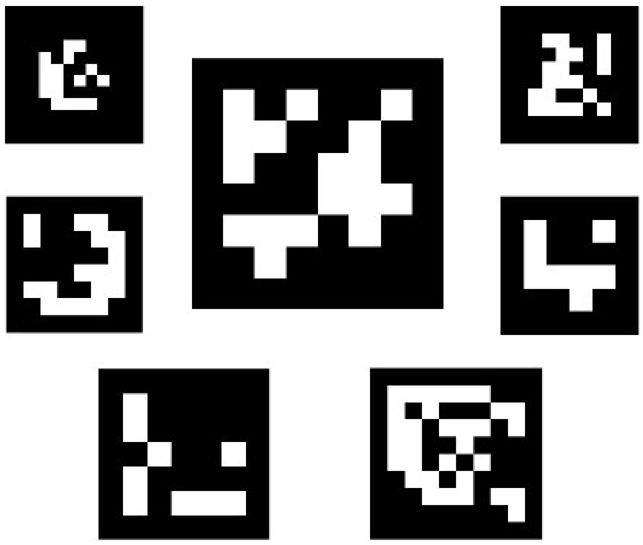
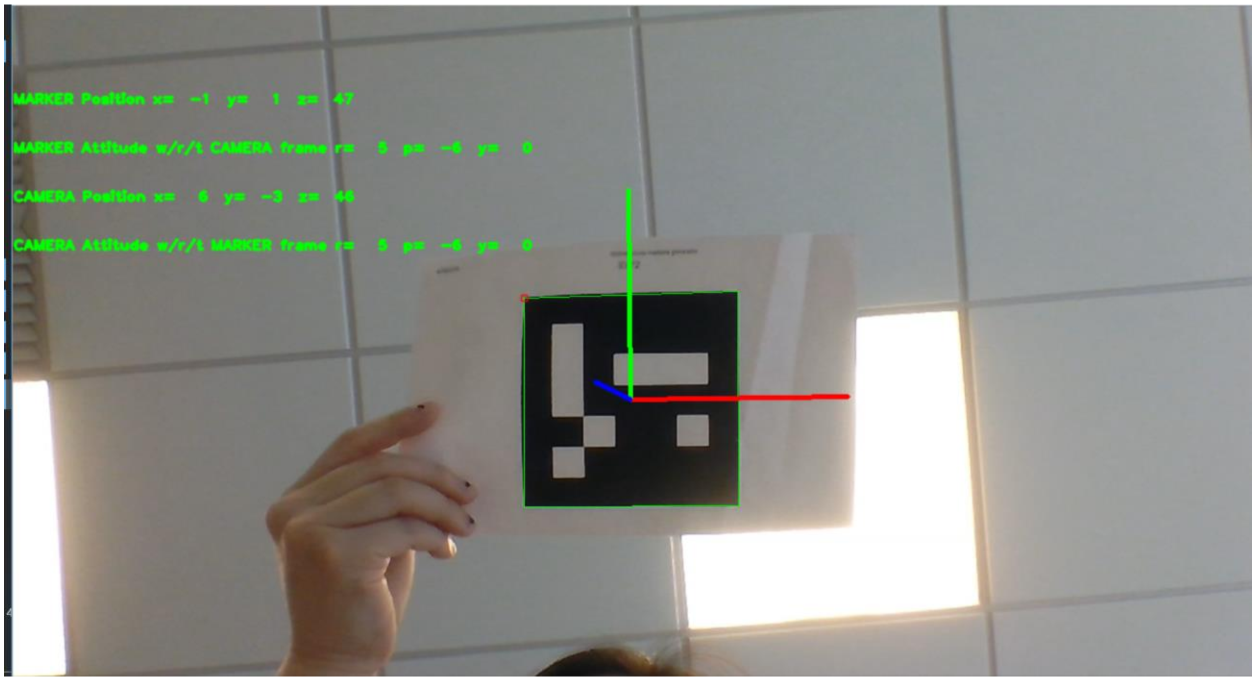


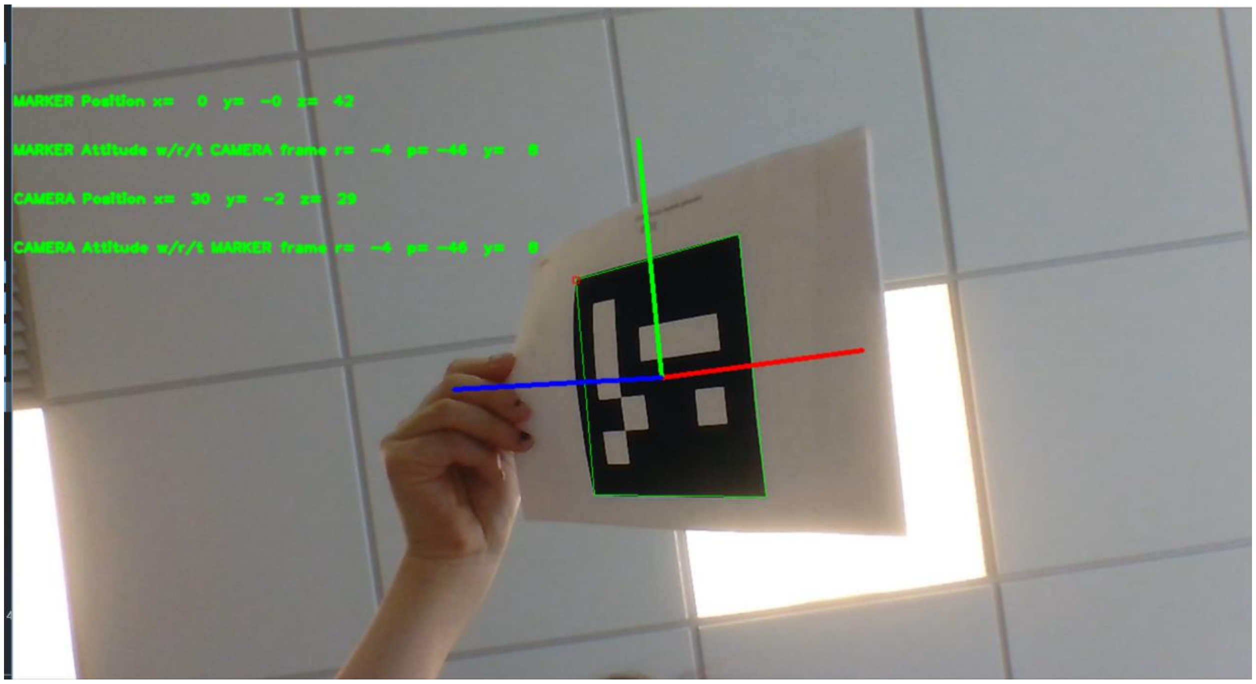
Figure 37: Sample ArUco Tags

In the program, we will have to specify the ArUco tag ID number and size (in mm) for the camera to recognize. Tags can be generated from an online generator (such as http://chev.me/arucogen/). A single ArUco tag is able to tell x, y, z distance from tag to camera. From the rotation matrix it returns, we are able to convert the rotation matrix to Euler angles to compute roll, pitch, yaw angles from camera to tag and vice versa.



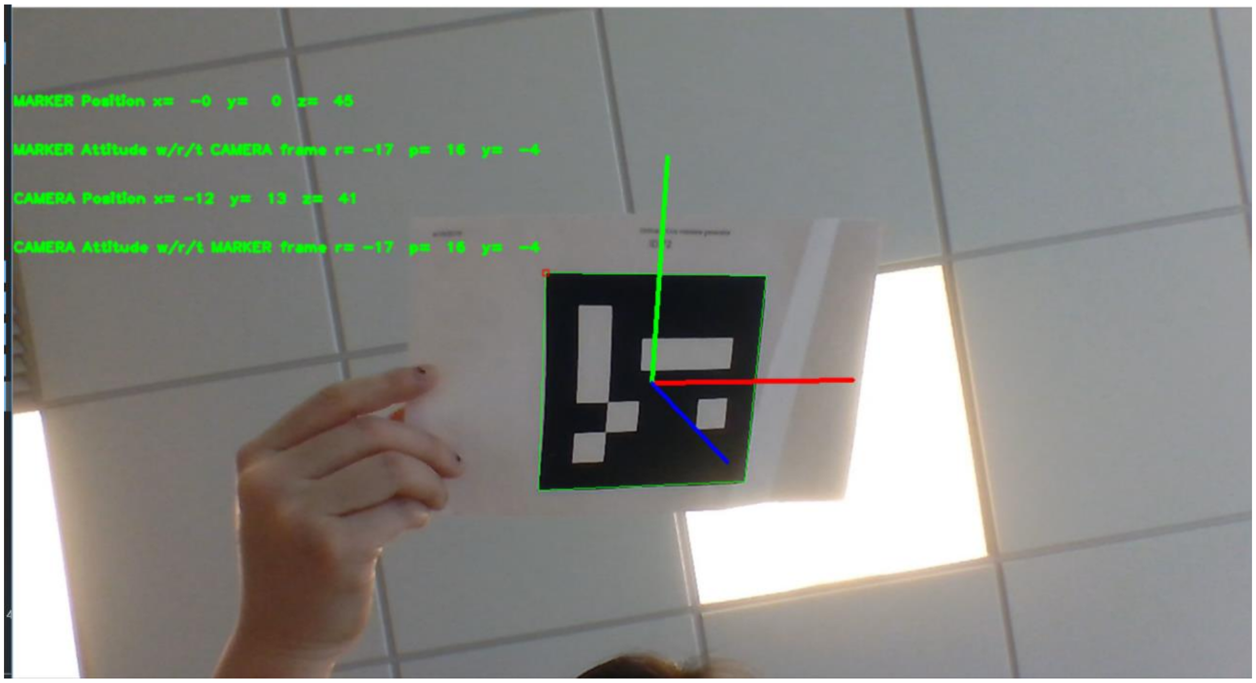
```
MARKER Position x=  -1  y=   1  z=  47

MARKER Attitude w/r/t CAMERA frame r=   5  p=  -6  y=   0

CAMERA Position x=   6  y=  -3  z=  46

CAMERA Attitude w/r/t MARKER frame r=   5  p=  -6  y=   0
```

Figure 38: Example 1 of Our Computer Vision with ArUCo Tag



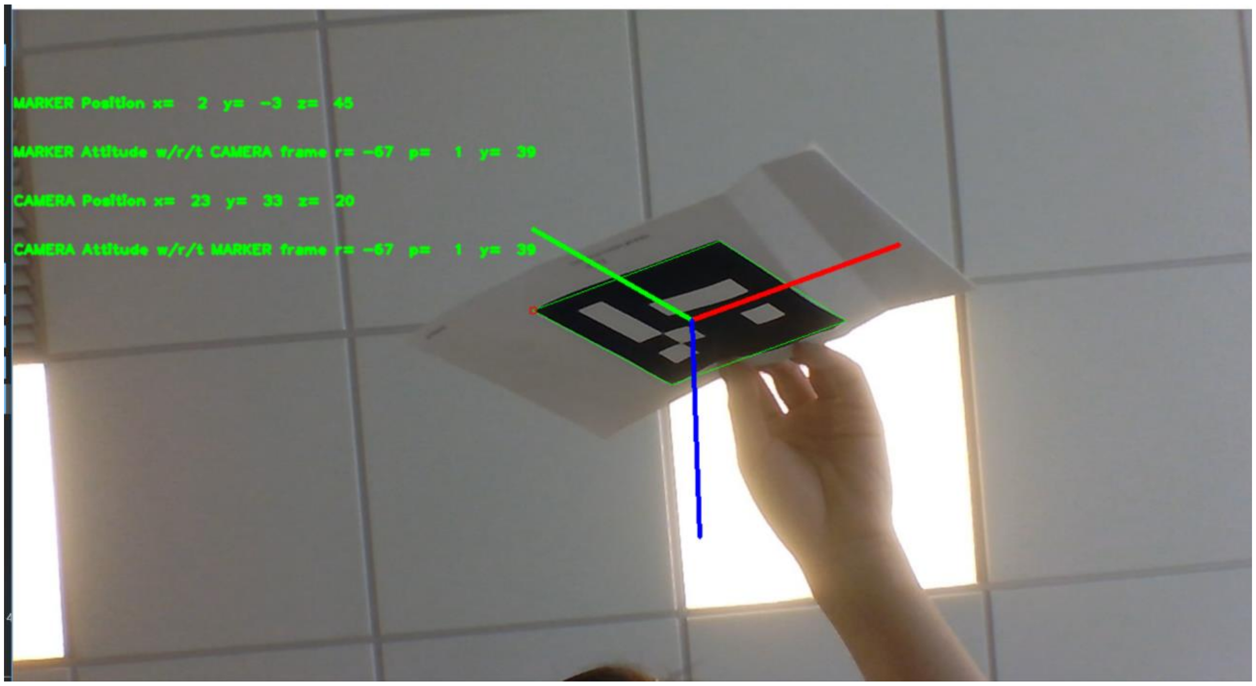
```
MARKER Position x=   0  y=  -0  z=  42

MARKER Attitude w/r/t CAMERA frame r=  -4  p= -46  y=   8

CAMERA Position x=  30  y=  -2  z=  29

CAMERA Attitude w/r/t MARKER frame r=  -4  p= -46  y=   8
```

Figure 39: Example 2 of Our Computer Vision with ArUCo Tag



```
MARKER Position x=  -0  y=   0  z=  45

MARKER Attitude w/r/t CAMERA frame r= -17  p=  16  y=  -4

CAMERA Position x= -12  y=  13  z=  41

CAMERA Attitude w/r/t MARKER frame r= -17  p=  16  y=  -4
```

Figure 40: Example 3 of Our Computer Vision with ArUCo Tag



```
MARKER Position x=   2  y=  -3  z=  45

MARKER Attitude w/r/t CAMERA frame r= -67  p=   1  y=  39

CAMERA Position x=  23  y=  33  z=  20

CAMERA Attitude w/r/t MARKER frame r= -67  p=   1  y=  39
```

Figure 41: Example 4 of Our Computer Vision with ArUCo Tag

Robot rotating angle will be computed by adding the servo angle to the ArUco's pitch angle. Angles are computed differently for left and right camera, depending on which direction the robot faces relative to the front of the field.

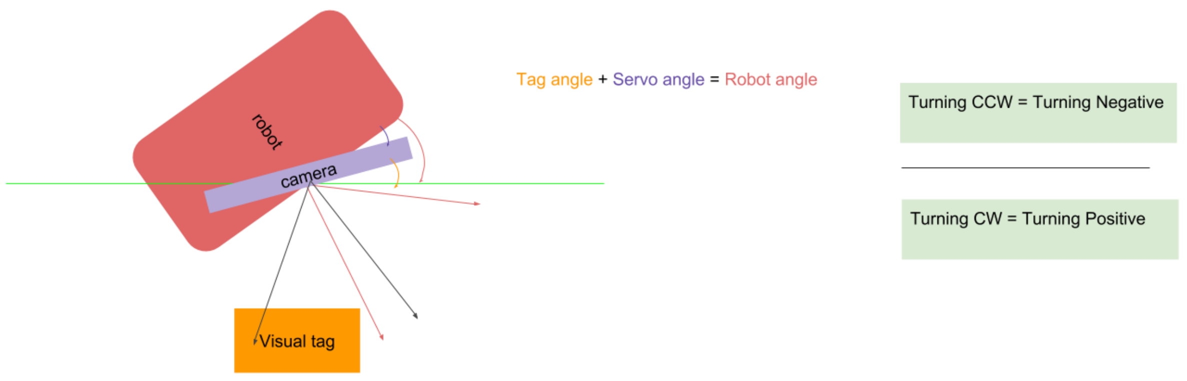

robot
camera
Visual tag
Tag angle + Servo angle = Robot angle
Turning CCW = Turning Negative
Turning CW = Turning Positive

Figure 42: Computation of Our Vision System

Visual tag returns an offset angle from the camera to the horizontal green line. Tag angle and servo angle can be both positive and negative depending on its pose. Therefore when they are added, they will return the true robot angle.

5.3.5 PID Closed-Loop Control

PID closed-loop concept is implemented for most feedback devices we have in order to increase accuracy for the automated behaviors. We used PID control for our navX and encoders to ensure that robot rotates and drives accordingly as commanded. Errors are fed back to sensors to make sure the robot behaves more accurately.

Chapter 6: R.O.V.E.'s Teleoperation

On top of autonomy, R.O.V.E is equipped with teleoperation for two purposes. Teleoperation assists us on testing different functionalities of the old and new system to make sure everything runs steadily before putting the robot into autonomous mode. Secondly, if anything happens during the autonomous run, the robot can always be controlled manually.

The robot can be controlled manually with the a game controller plugged into our laptop. Without pressing any button, left and right joystick's Y-axis will control robot drive motor. With left stick in charge of speed and right stick in charge of rotational angle. When *A button* is pressed and held, right joystick Y-axis controls the *extender*'s motor. When *B button* is pressed and held, right joystick Y-axis controls the *scoop's* motor. When *X button* is pressed and held, right joystick Y-axis controls the *dump's* motor. Since Y-axis value from -1 to 1 from North to South direction, we flipped the value of Y-axis so that robot motor runs forward when joystick is pressed to upwards.

Chapter 7: R.O.V.E.'s Dust-Proof Design

Our team plans to implement all electronics in a custom made box using the material polypropylene. This design will be a two shelved box encompassing all electronics in which wire connectors will be hot glue gunned through wire holes. This will allow connection from the inside and outside of the box without allowing large openings for dust to enter the actual box. Side holes are patterned on the side of the box in effort to allow some air to flow through the box while mitigating any dust to travel vertically into the electronics. The wire connectors will connect to all electronics and motors on the outside. This box will be fastened with machine screws to allow ease of access to the electronics.

Chapter 8: R.O.V.E.'s Performance Evaluation and Optimization

The new designs were constantly tested during developing process, whether or not on the whole robot, in order to make adjustments on time.

8.1 New Digging Mechanism Testing

The process to test for potential upgrades was used consistently throughout the design process to reduce possible unknown variables limited by the software and operation of the robot.

8.1.1 Method of Testing

To test during initial analysis, in addition to testing upgrades, our team had directly controlled individual motors and only operated a singular motor one at a time. Upon connecting the designated motor controller - Talon SRX, we powered the motor using the teleoperated controller to regulate motor speed and power distribution. This was performed for the digging chain system and the tape drive extender. The analysis began with multiple scoops attached, then procedurally remove scoops until only one remained. All following testing was performed with only one scoop for the sake of reducing any variety in scoop design to test the chain guide.

For the digging system, the scoops would run under no load, to determine any possible issues with the design. The scoops were driven under the minimum of 2 Amperes to minimize any potential damage that may be caused during operation. This procedure of operating the motor is the same for the tape drive motor. To allow the robot to operate with its excavation system, the wooden frame was built to operate the robot without load, while suspended. This enables the robot to be viewed at full extended length and retracted without specific set up for each configuration, while allowing possible modifications at the same time.

8.2 Software

There were multiple methods we used to test the software without requiring the robot to be fully constructed.

8.2.1 Computer Vision

Computer Vision can be tested separately from the robot. Cameras are mounted on the servos when tested. One camera/servo is run first and if it cannot find the goal image, second camera/servo will then run. Once the visual tag is found, the angle and distance values from tag to camera are sent to NetworkTables. Raspberry Pi and roboRIO worked great on communicating with each other and with our laptop. Through testing, we were able to add some time delay of 3 seconds between rotations so that camera can capture the visual tag.

8.2.2 Autonomous Routines

Before running the entire ten minutes of autonomy on the 150lb-robot, we ran the code on Talon SRX. With the green/red LED signals to make sure that timing is correct and motors are only powered when it is set. By doing so, we were able to find some loopholes in the program logic and fixed them. In addition, one thing the state machine was missing is setting the output value of the running motor controller to 0 before switching to the next state. There was difficulty where even though the drivetrain speed and rotational rate are set to 0, its Talon motor controller was still powered. Through debugging process, we found out that we needed to disable navX PID controller along with zeroing out the motor power. NavX was not called at any point but once its PID was enabled but it was always in active.

8.3 Autonomy

Before the robot is sent into a complete autonomy test mode, we had to carry out some manual runs to calibrate some data to make sure the robot runs smoothly without damaging itself. A log system was designed where the program will save a CSV file on time vs scoop current draw for us to determine what the current draw for when the scoops dig sand or gravel, and when the scoops don’t interact with anything. We are also export percentage output data from each Talon SRX to motor at each state when running manually to ensure that the robot don’t rotate or dig too fast or too slow when it performs autonomy.

Chapter 9: R.O.V.E's Accessibility For Later Teams

One of the biggest factors that set us back at the beginning of the development process was that there was no instruction on how to run the robot. There was no guidelines on how to wire the hardware and power them. There was no documentation on where to start on the code while the robot program is considerably complicated. Lastly, there was no way to disassemble the robot without having to lift the robot up in the air and extend the digger by running the extender motor. We think it is essential to document the instructions since this project is passed on through different generations.

9.1 Disassembly Guide (Excavator and Motor Mount)

To allow ease of use in the future during this project and potential future iterations, our team took into consideration of designing our robot to not only solve the stated issues but to make it easier to modify in the future. To remove the excavator plates, simply unscrew the three screws attached to the tape drive connector on each plate. Then, as you extend the plate using the linear drawer, unhinge the plastic clip connecting the two halves of the linear guide. This will allow access to modify the side plates, chain guide, or even the chain itself in the chain guide.

All sprockets and sprocket hubs are standard designs, therefore if the chain is to be placed, the only concern should be lining the tabs to be parallel to one another. In addition, the sprockets have shaft collars that are accessible to remove the shaft from the motor mount. All machine screws have hex caps.

9.2 Control System Guide

A readme file was written to not only guide later teams how to run the robot program but also help them navigate through the code. There are instructions on what files along with their functions are in different directories. Each method and variable is named clearly and commented for readability.

Electronically, we are providing an easier-to-understand diagram to make sure if the electronics are unwired, the team will still know which port and which power each element requires to put them back together and run them.

Chapter 10: Future Work

We have a lot of ideas but being a two-person team restraints us to accomplish some of them.

10.1 Mechanical Design

A potential design in the future would be to implement a excavating system that provides support from the bottom or rear end of the scoop. Potentially with a band similar to treadmill belt, could support the scoops from the bottom to reduce cantilever forces in the scoop system itself. Furthermore, a redesign to a new excavating system to reduce compressive forces the robot undergoes during mining may be a potential iteration. These two ideas in conjunction could present a new excavating system similar to bucket dredging systems used in today's seabed mining.

10.2 Electrical Design

Although our robot’s bucket is quite big, as the robot is getting more efficient, we think it is better to implement a sensor to determine the capacity of the materials that the bucket is holding during mining process. Right now, we are assuming that the robot can dig less than its capacity within the time span of the competition. However, we are limiting our robot with this assumption. Having a system that can detect how much the robot is actually holding will allow robot to expand its capacity, run faster and run multiple trips. An infrared sensor can be used to perform this task.

The second improvement is on the camera servos. The cameras we have are regular clipping cameras. To mount it on the servos, we disassembled and removed more than half of the original cameras. The way the cameras is mounted on the servos is not stable and cameras will drift over time as servo turns. There are some options. First option is to create a 3D printed box for the camera onto the servo. Team can also create a drill hole that fit the servo’s screw.

10.3 Software Design

Although MagicBot framework allows the program to be clean, run and tested easily, there is always space for development. The autonomy can be improved on both design and implementation aspect.

10.4 Testing Area and Robot Home

For future reference of potential testing, future teams should contact Worcester Sand & Gravel to create a sand pit with gravel. This would require a needed base of operations for the pit and robot to be tested in. Therefore, it is highly advised that potential teams locate a new site to store, work on, and test the robot.

Chapter 11: Conclusion

This project describes our process on upgrading WPI NASA mining robot from October 2018 to April 2019. Our goals were to redesign and implement the digging mechanism as well as enabling the robot to behave autonomously within the scope of the competition. Despite the cancellation of NASA RMC half way through the project, our team still accomplished many of the goals that were determined at the beginning of this project.

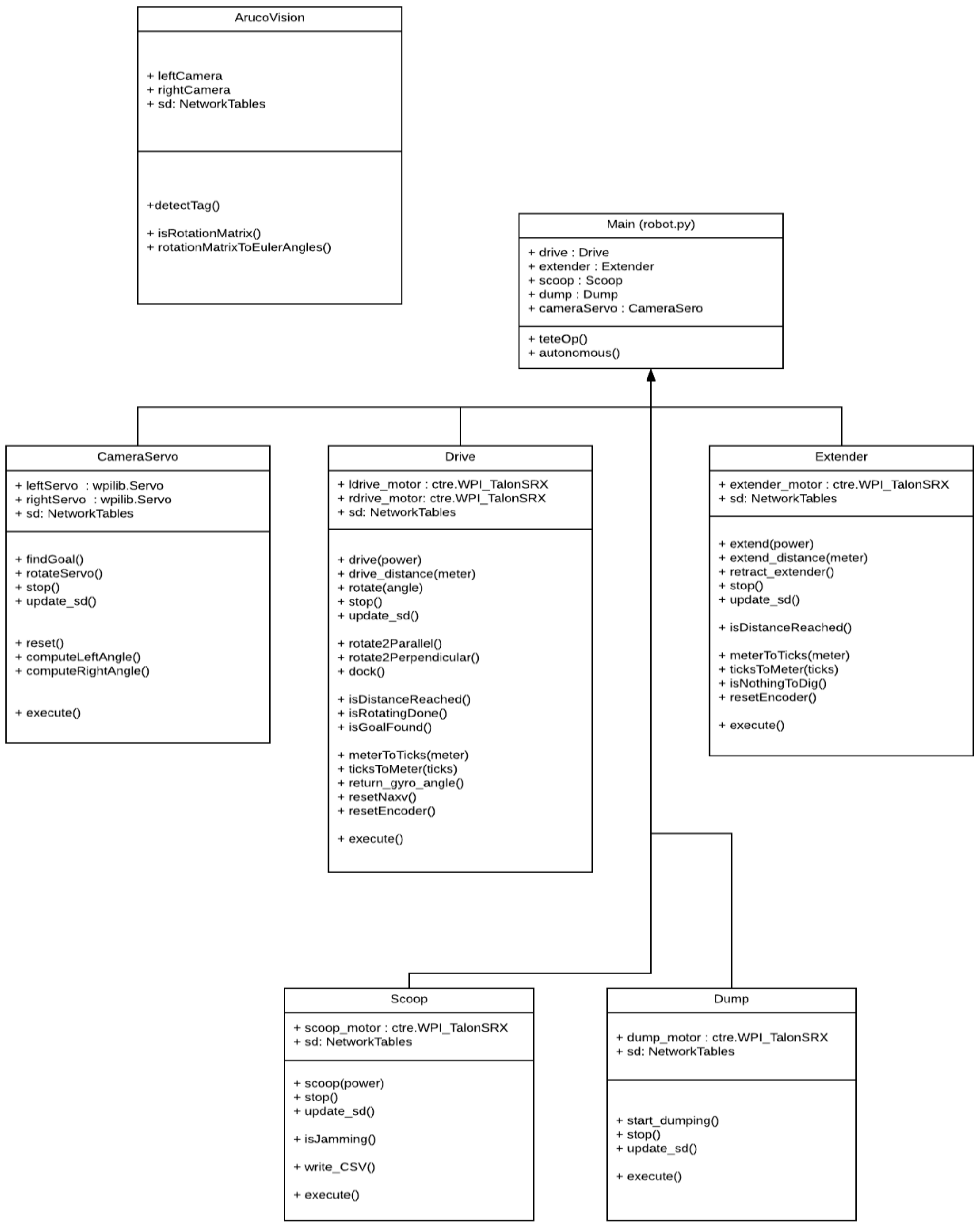
Table 2: End Progress Table

Parameter	Specification	Requirement Met
Maximum Size	1.5 m x 0.75 m x 0.75 m	Met
Maximum Mass	70kg	In progress of measuring
Digging Mechanism	Scoops turns smoothly when going through sprockets and run without jamming when fully emerged in sand and gravel	Designed and manufactured, not tested
Control System	Full Autonomy	Designed and tested separately with motor controllers but not with robot
Dust-proof	Design a box to protect the electronics from dust	In progress

At the time of submission, we are in the process of testing the new system with load. Our team does not have access to a pit of sand and gravel similar to the arena used during the competition. However, using excess gravel, we were able to test the new motor mounting plate design and its efficiency. The gravel carried in the scoop was easily able to drop into the bucket of the with the new geometry of the chain path.

We found that the tolerancing of the scoops, fastener tabs, chain guides, and chain are extremely tight and are the main cause of the forms of jamming that is incurred as we tested our new design. Ensuring the tabs of each scoop are dimensioned exactly will allow the scoops to travel through the path unaffected. We discovered this by testing the tabs connected to each other without scoops and with the scoops. This helped us understand that the chain system is not at fault, but the fastener tabs are too thick, forcing the chain to displace just enough to jam at the engagement corner but nowhere else in the system. The power transmission of the shaft and coupler are inefficient as well, leading to one side traveling slower due to the coupler not transmitting the power to the second side of the chain system.

Appendix 1: Robot Program UML Diagram



References

[1] NASA. (May 24th, 2019). *Programs and Missions*. Retrieved from https://mars.nasa.gov/programmissions/missions/missiontypes/rovers/

[2] Alabama Astrobotics. (May 24th, 2019). *The University of Alabama Astrobotics*. Retrieved from http://www.alabamaastrobotics.com/nasa-rmc-2018-cycle.html

[3] Castelino, Hagen, Khan, Kumar, and Patias. (May 24th, 2019). *Ibex: Robotics Mining Platform*. Retrieved from https://digitalcommons.wpi.edu/cgi/viewcontent.cgi?article=7622&context=mqp-all

[4] Quora. (May 24th, 2019). *What qualities or capacities of Python makes it a good programming language for robotic engineering*. Retrieved from https://www.quora.com/What-qualities-or-capabilities-of-python-makes-it-a-good-programming-language-for-robotic-engineering

[5] Quora. (May 24th, 2019). *Why is Robot Operating System ROS preferred*. Retrieved from https://www.quora.com/Why-is-Robot-Operating-System-ROS-preferred

[6] Intorobotics. Quora. (May 24th, 2019). *15 reasons to use the robot operating system ROS*. Retrieved from https://www.intorobotics.com/15-reasons-to-use-the-robot-operating-system-ros/

[7] BaneBots. (May 24th, 2019). *P60 Gearbox, Custom 256:1*. Retrieved from http://www.banebots.com/product/P60C-4444.html

[8] VEX Robotics. (May 24th, 2019). *775Pro.* Retrieved from https://www.vexrobotics.com/775pro.html

[9] VEX Robotics. (May 24th, 2019). *VersaPlanetary Gearbox.* Retrieved from https://www.vexrobotics.com/versaplanetary.html#Docs_&_Downloads

[10] McMaster-Carr. (May 24th, 2019). *Chain Guides.* Retrieved from https://www.mcmaster.com/chain-guides

[11] National Instruments. (May 24th, 2019). *roboRIO Advanced Robotics Controller.* Retrieved from http://www.ni.com/en-us/shop/select/roborio-advanced-robotics-controller

[12] Raspberry Pi. (May 24th, 2019). *Raspberry Pi 3 Model B+*. Retrieved from https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/#buy-now-modal

[13] Kauai Labs. (May 24th, 2019). *navX-MXP Robotics Navigation Sensors.* Retrieved from https://www.kauailabs.com/store/index.php?route=product/product&product_id=56

[14] Robot Py. (May 14th, 2019). *RobotPy*. Retrieved from https://robotpy.readthedocs.io/