Intelligent Robot Design for VEX U Skills Challenge "Change Up"

Misael Marquez, Mechatronics Engineering
Vaughn College of Aeronautics and Technology, NY, USA
misael.marquez@vaughn.edu

Abstract - This project outlines the design, build, and program of an autonomous robot. This robot will pick up and hold a maximum of three 6.3-inch diameter plastic hollow balls and place them into a basket-like goal to score points. The robot controls are automated using infrared sensors, encoders, optical sensors, and distance sensors. A de-scoring mechanism will use rubber and custom "flap" wheels to provide enough torque to grab the ball from the goal and remove it. The scoring mechanism includes rubber band rollers that will index and control the balls within the robot. To minimize the weight of the robot, components of the drivetrain and conveyor will be 3D printed. This report will display the process of selecting a de-scoring mechanism, scoring mechanism, drivetrain, and their respective programs.

Keywords: Autonomous, Robot, Smart, Feedback

I. INTRODUCTION

Every year Vex Robotics releases a new game in which several teams compete against each other. This year, the game is coined "Change Up." Two separate competitions exist with this game. This report will be based on the skills competition. The skills competition is where a robot is alone on the field (Fig. 1) and has one minute to manipulate the game elements in goals to achieve the highest score possible. Points are given by the number of balls placed in goals and rows owned by an alliance. Every ball in each goal is worth one point to the respective colored alliance. The top ball in each goal decides who gains ownership of said goal. A row created from owned goals is worth six points. There is a total of nine goals with one goal in each corner of the 12-foot by 12-foot field. There is also one goal at the center of all the side panels and one in the center of the field. The center goal is the most difficult to score and de-score. Each robot must start in a home zone but must not contact outside of it at the beginning of the match.

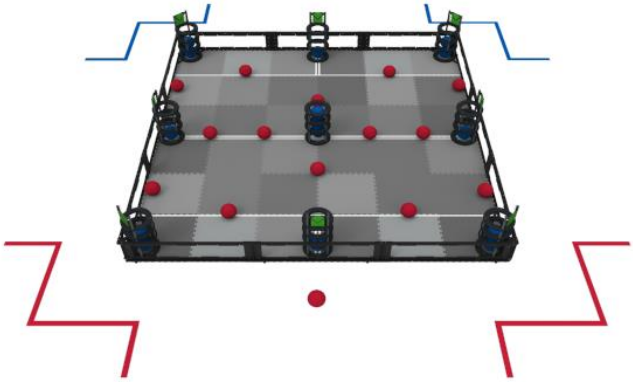

Figure 1: Change Up Skills Field Layout [1]

Digital Object Identifier (DOI):
http://dx.doi.org/10.18687/LACCEI2021.1.1.644
ISBN: 978-958-52071-8-9 ISSN: 2414-6390

A. Objective

The objective of this project is to design, build, and program a robot that can move around the field quickly, while also collecting balls and then later placing and removing them from goals. To accomplish the first goal, a light and fast drivetrain is constructed. The second goal is accomplished by creating an actuated set of intakes to collect balls. These balls are fed through a conveyor system that places and sorts the balls by color.

B. Related background

Drivetrain: A drivetrain is a subsystem that controls the robot's positioning on the field. This consists of a chassis on which the other subsystems mount onto. The chassis must be sound enough to support the weight of the robot's other subsystems while transferring the weight over wheels.

Actuated Intakes: An intake is a subsystem that collects objects. The intakes used on this robot consist of rubber paddles and rubber wheels parallel to one another that spin in the same direction. The set of intakes are placed such that they compress the balls when collecting and removing them from goals. Actuated intakes allow the intakes to open and close to increase the room for error in the autonomous routines when approaching balls.

Conveyor: A conveyor is a subsystem that moves the balls up and down the robot. The conveyor consists of a set of four rubber band wheels that compress the balls into a backboard and hatch. The hatch is used for sorting purposes by opening the back of the conveyor and letting a ball exit without being scored.

II. ENGINEERING REQUIREMENTS

Compact – The robot has a sizing requirement of fitting in a 15-inch cube at the start of the run. Once the run begins, the robot may expand to any shape desired.
Light – The robot must be light to reduce the resistance applied onto the motors that propel the robot forward.
Efficient – The robot's functionalities must solely be used to manipulate balls, or to reposition the robot on the field.

III. HARDWARE DESIGN

A.1 Drivetrain

The drivetrain used was a 13.5-inch by 15-inch tank configurated chassis with six motors. This configuration has two wheels in the front and back of the robot. The six 200 rotations per minute motors were placed on the rear of the robot.

Each side of the drivetrain (Fig. 2) had three motors with their inputs combined via a 1:1 gear train to the output shaft. The output shaft has a sprocket which in turn transmits power to the wheel in the front. The output shaft transmits power directly to the rear wheel. All four wheels were 4-inch Omni-directional wheels. These wheels were used as they allow movement in all directions which decreases friction applied to the wheels when turning. The connection from each wheel and sprocket was made tight using special 3D printed spacers. These spacers fit in between the wheel spokes and were attached to the sprocket via pre-existing holes. This reduced the backlash between the sprocket and wheel, thus increasing the robot's control of its position.

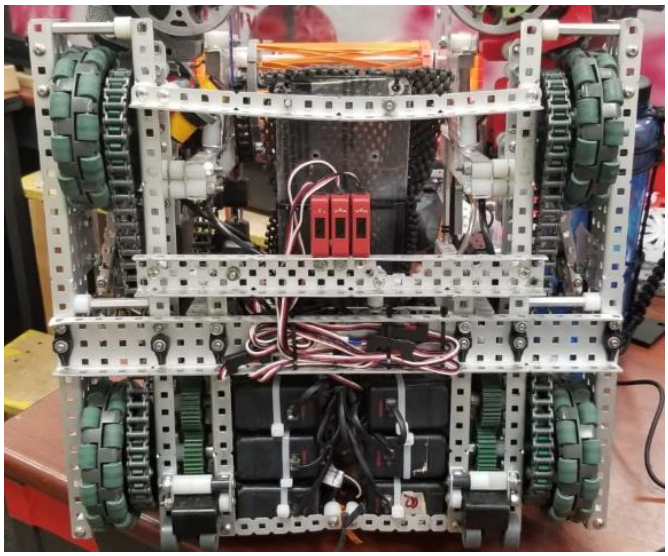

Figure 2: Tank Configurated Drivetrain

A.2 3D parts

Most of the robot's weight came from the drivetrain. To minimize weight, various sections of the drivetrain were 3D printed in polylactic acid (PLA). When making the parts, smaller components were added onto them to reduce the overall part count. The main supports of the drivetrain remained as 1.5-inch by 0.5-inch c-channels. The parts replaced the corner portions of the drivetrain and held the wheels.

B.1 Intake Mechanism

The intakes were initially composed of standard 0.5-inch by 1-inch, and 0.5-inch by 1.5-inch aluminum c-channels . The 4-inch outer diameter VexPRO flex wheels were mounted onto the c-channels and were driven together via a gear train. The 600 rotations per minute motor indirectly powered the intakes via the gear train leaving a speed ratio of 1:1 from the motor to the rear wheel and 1:1.67 from the motor to the front wheel. The mounts created for the intakes allowed them to fold upwards. This folding movement aids in the robot's 15-inch compression. Fig. 3 displays the right intake with the motor offset from the line of action of the wheels. If the motors were on the line of action of the intakes, the motors would end up hitting the bottoms of the goals. This action could damage the motors or goals after excessive use.

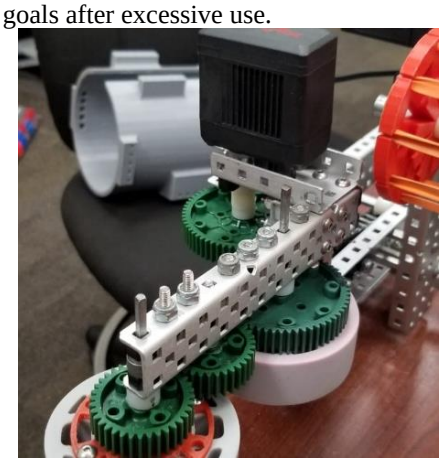

Figure 3: Intakes

The intakes were hard mounted, which were causing issues. The intakes must compress the ball when collecting them to properly have control of the balls and when pulling them out of goals. If the intakes are too close, then intaking balls around the field would be very difficult and the ball would get pushed to another area of the field. To resolve this, the mounts were adjusted such that they can pivot. The pivoting action is pneumatically controlled by a single action piston. The piston is attached to the pivoting intake mounts via a linkage such that the piston's initial position is when the intakes are closed.

B.2 3D Parts

The original intake wheels only had one point of support, which was very problematic. A second support at the bottom of the intake is required to keep the wheels straight and gears in mesh. A 3D part was created on SolidWorks using the spacing requirements for the part. The part was printed using PLA and carbon fiber composites. Due to printing capabilities, the part was printed solely out of PLA (Fig. 4).

Figure 4: 3D Printed Intake Wheel Holder

The pivoting mounts (Fig. 5) proved to be too weak to support the weight of the intakes. Much like the intakes, the mounts were redesigned on Solidworks. The mounts were then printed in PLA with several iterations.

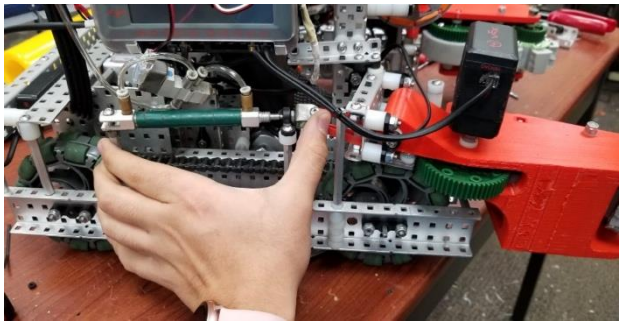

Figure 5: Pneumatically Controlled Intakes

C.1 Conveyor

The conveyor system (Fig. 6) was constructed from rubber band rollers on c-channel uprights. The bottom half of the conveyor was mounted off the 0.5-inch by 1-inch c-channel uprights because the thickness of the c-channel did not allow the intakes to fold completely within the robot. The top half of the conveyor was constructed from 0.5-inch by 1.5-inch c-channels directly above the bottom conveyor assembly. Fig. 7 shows the pivot point that connects both halves of the conveyor. Both Conveyor assemblies were powered by a 600 rotation per minute motor. The top conveyor was directly powered, and the bottom conveyor was powered indirectly with a 1:1 speed ratio to allow the intakes to fold up.

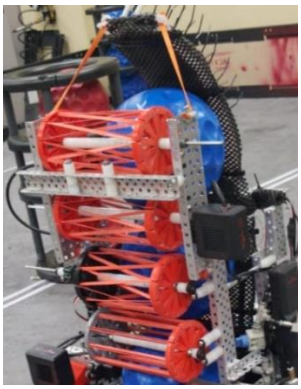

Figure 6: Rubber Band Roller Conveyor

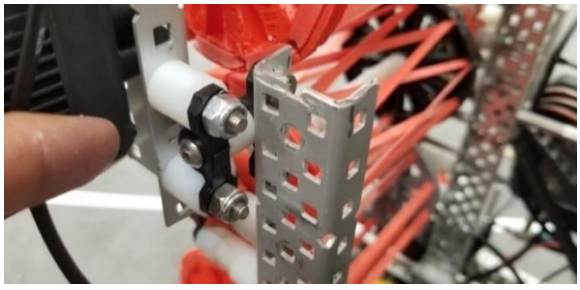

Figure 7: Conveyor Hinge

A hatch on the rear of the conveyor (Fig. 8) was implemented for sorting purposes. When the double-action piston extends, the hatch closes, letting the balls move upwards to be scored. When the double-action piston is retracted, the hatch opens, releasing the ball towards the rear of the robot.

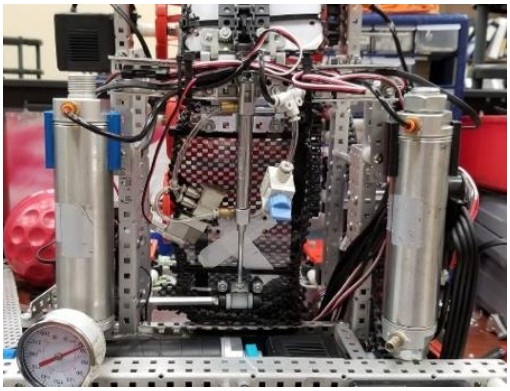

Figure 8: Hatch

C.2 3D Parts

The conveyor system had many components. Each component had a probability of failure and with more components on a given system, the overall probability that the assembly would fail is high. Failure can come in the form of a bolt coming loose, or a part breaking. Replacing a part would be extremely difficult and tedious. Therefore, the individual conveyor assemblies were minimized in the sense that one part (Fig. 9) took the place of several. This part was very easy to replace in case of failure. This part also reduced the weight of the robot as the aluminum and stainless-steel components were replaced for a thicker PLA version.

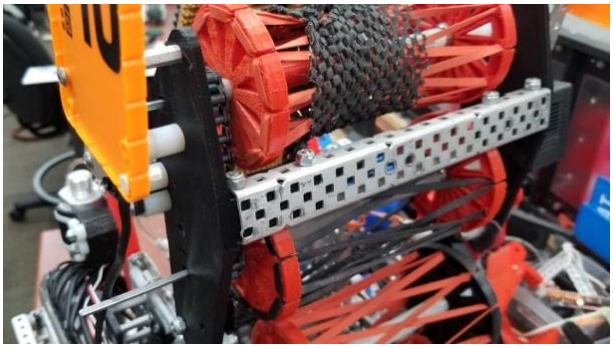

Figure 9: Conveyor With 3D Printed Parts

IV. AUTOMATION

A. Sensors

Automation of the various subsystems was done through the usage of sensor inputs. The use of sensors is very important as they are the interface between the robot and its program. They help the robot make precise movements based on each sensor's reading as opposed to time delays.

A.1 Line Sensor

The line sensor [2] is an analog infrared sensor. The sensor has an infrared LED and infrared light sensor. The line sensor sends a signal away from itself and using the light sensor it detects the reflectivity of a surface. Different colored surfaces reflect differently. Lighter colors will make the line sensor return a low value while darker colors return a higher value. Since there are lines throughout the field, the robot can use the line sensor to adjust its path. This sensor is placed at the bottom of the drivetrain of the robot. Initially, there were three-line sensors in an array such that the robot can follow a line. While the robot could follow a line, it was difficult for the robot to follow the line quickly. Eventually, two sensors were removed, only keeping one to tell the robot when to stop.

A.2 Infrared Sensor

The infrared sensor works in the same manner as the line sensor. The sensor has an infrared LED and infrared light sensor. The infrared sensor sends a signal outward and detects when it returns. The signal only returns when there is an object in range. The range of the sensor is adjusted via a flat head screw near the end. The sensor then returns a digital value of one to the V5 Brain. Two infrared sensors are used on the robot, one on the intake, and one on the hatch. These are used to detect when a ball is within range to feed the conveyor and to tell the hatch when to close.

A.3 Inertial Measurement Unit

The inertial measurement unit (IMU) [3] is used to return and control the heading of the robot in between straight-line movements. The IMU has a 3-axis accelerometer and gyroscope. The heading is returned in values of zero to 360 degrees. The IMU must be calibrated before every run as it achieves an initial heading after calibration in which all other headings are referenced off.

A.4 Encoder

The encoder is built into each of the motors used on the robot. The encoders measure the rotation of anything mounted onto the output shaft of the motors. Using the values returned from the encoder and conversions, the robot can calculate its position. The number of degrees (or ticks) that the encoder must count before reaching its target distance is given by the conversion "(1)" where the length is the distance the robot must travel. The encoders are also used in rotations to help the robot rotate about its center of mass. The user should reset the encoder in between different movements as the encoders save their previous value.

$$\text{Encoder Ticks} = (\text{length}*360)/(\pi*\text{Wheel Diameter}) \quad (1)$$

A.5 Distance Sensor

The distance sensor [4] uses a class 1 laser and reads the distance to an object. The laser is very narrow and can only read up to two meters. The distance sensor works similarly to the infrared sensor in the sense that its range is adjustable, but the range is adjusted electronically. The distance sensor also returns a value in inches or millimeters. This sensor is used to detect the presence of a ball in three locations in the conveyor system, and the robot's distance to the walls of the field.

A.6 Optical Sensor

The optical sensor [5] detects the color of the object in front of it. This sensor also has a white LED light to aid in color detection. This sensor is used to sort the balls to either release them at the hatch of the conveyor or to feed them upwards such that the robot may score it.

B. Programs

The robot is programmed in C++ with an application called "VexCode Pro." This application was created by Vex to interface with their products. External libraries were not required for the robot's programming. The program that runs the robot is split up into various functions, tasks, and classes. A block diagram of a skills routine is shown in Fig. 10.

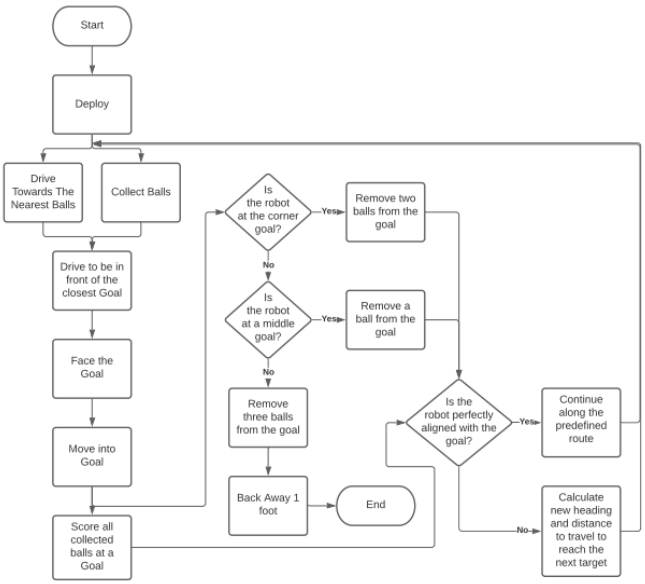

Figure 10: Block Diagram of Autonomous Routine

B.1 Robot Setup

Each motor subsystem is placed in a function where a value can be passed through as a parameter which is then set as the motor's speed. This is also repeated for the pneumatically controlled subsystems. These were created to minimize the readable size and repetitive lines in the program. An example of these functions is shown in Fig. 11.

```
void runIntake(int speed)
{run(RR,speed);run(LR,speed);}

void openClaw()
{Claw = 1;ExtClaw = 1;}
```

Figure 11: Actuated Intake Subsystem Control Functions

B.2 Movement Functions

A PD closed feedback loop (Fig. 12) is used to control the robot's exact positioning. Two functions use the encoders and the inertial measurement unit as feedback for the straight-line and rotational movements, respectively.

```
while(true)
{
  pidError = EncoderTicks - enc(RF);
  if(abs(pidError)<2)
  {
    StopDrive(hold);
    pidError = 0;pidErrorprev = 0;
    pidIntegral = 0;pidDerivative = 0;
    wait(100);
    break;
  }
  pidIntegral = pidIntegral + pidError;
  pidDerivative = pidError - pidErrorprev;
  pidErrorprev = pidError;
  pidDrive = (kp*pidError)+(ki*pidIntegral)+(kd*pidDerivative);
  if(pidDrive > MaxSpeed){pidDrive = MaxSpeed;}
  if(pidDrive < MinSpeed){pidDrive = MinSpeed;}
  leftDrive(pidDrive);
  rightDrive(pidDrive);
}
```

Figure 12: PD Control Feedback Loop

A PD loop is used because any value given to the integral gain would cause the drivetrain to heavily oscillate. A PD loop is a subsidiary of a PID loop. A PID loop is a method of control that is proportional in some manner to the error of a system. This system is a drivetrain. PID stands for Proportional, Integral, Derivative. These describe the different ways that the error of a system can be manipulated and returned as feed back to the system used. There are three gains that are multiplied to achieve and output, kP, kI, kD. These are multiplied to the error, integral of the error, and derivative of the error, respectively. These values are then combined to achieve the output as shown in Fig. 13. The proportional and derivative gains were selected via an experimental process. This process commences by having all gains other than the proportional set to zero. Then the proportional gain is set to a very small value. The value is doubled until the robot overshoots its target distance, afterwards it is halved. The derivative gain is found by setting the derivative gain to a small value, then increasing the value until the robot oscillates. Once the robot oscillates, the value is halved. These values give the general tunings for the robot.

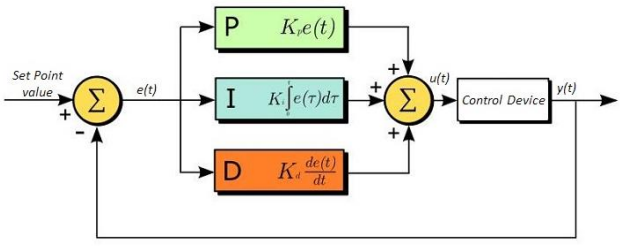

Figure 13: Illustration of PID Feedback Loop

An inertial measurement unit drifts in value over time. This can come from poor calibration or the rise in temperature when there is power directed to it. Not all inertial measurement units drift in the same direction or by the same amount. To counteract the drift given by one IMU, two IMUs are employed. The robot takes the average of the values returned by the inertial measurement units and uses them for turns.

Within the movement function is a condition that checks if a certain distance marked by the user as a parameter is met, and if so, the function would then start a task (Fig. 14). These tasks control the position of the balls within the robot based on the readings of the distance sensors on the conveyor. Since the balls are very light, weighing 168g, and the conveyor has a small moment of inertia, the motors controlling the ball's position do not require a feedback control loop.

```
if (((abs(ToWall.objectDistance(inches)) \
   < (abs(startDistance)+5)) &&\
   (abs(ToWall.objectDistance(inches)) \
   > (abs(startDistance)-5))) &&\
   (taskStarted == false))
{
  ///////// START SECOND TASK/////////
  taskStarted = true;

  if     (secondTask == "INTAKE")
  {vex::task BI(BallIntake,15);}

  else if(secondTask == "SPIT")
  {vex::task BS(BallSpit,15);}

  else if(secondTask == "EJECT")
  {vex::task BE(BallEject,15);}

  else if(secondTask == "ADJUSTSCORE")
  {vex::task BAS(BallAdjustScore,15);}
}
```

Figure 14: Start Secondary Task Condition and Selection

B.3 Trigonometry

The robot's position may have a small error that comes from each movement, but these errors build up over time. While this may not affect the robot's ability to go from the first goal to the second, it does affect the robot's ability to reach the third. To counteract this, the robot's position is expected to have errors. The robot will then be able to correct for it using a series of trigonometric calculations to recalculate a new path. While this process does correct for its scalar position, its exact position along with its heading will not be ideal. However multiple instances of this error feedback allow the robot to swiftly move from one goal to another.

The trigonometry portion works through connecting two straight line movements with their resultant. In any case, the resultant of where it is to where it needs to be is constant. using this, the robot can tell if it is going to move perfectly or not. Using the onboard IMU, the robot reads its current heading and compares it to what its heading is expected to be. If the headings are the same, it will continue along its predefined route and makes up the "Triangle of perfect movement." If the headings are not the same, then it will make up the "Triangle of error movement" as shown in Fig. 15. In this case, the robot's first movement will be of the same magnitude as if it were part of the perfect movement.

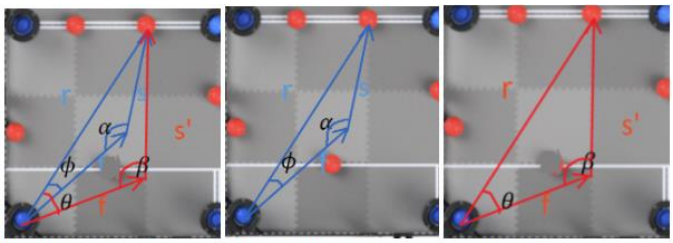

Figure 15: Illustration of Perfect vs Error Movements

The new heading and second movement will be recalculated using the law of cosines and law of sines.

$$s'^2 = f^2 + r^2 - (2 \times f \times r \times \cos(\theta)) \quad (2)$$

$$r^2 = f^2 + s'^2 - (2 \times f \times s' \times \cos(\beta)) \quad (3)$$

By manipulating "(2)" and "(3)", an equation for s' and β are attained respectively:

$$s' = \sqrt{f^2 + r^2 - (2 \times f \times r \times \cos(\theta))} \quad (4)$$

$$\beta = cos^{-1}\left(\frac{r^2 - b^2 - f^2}{-2bf}\right) \quad (5)$$

Each goal is given an object in the calculation class. For each object, two straight-line movement lengths, a perfect approach heading, and the resultant of the two movements are passed into the class's variables. The straight-line movements can be any combination of forward or backward movements. The end points of the movements are where the robot's center of rotation lies. Using these values, a function in the class calculates what the new heading and moving distance are such that the robot will end at its desired position. The creation of this compensator was created based on the thought that the distance between balls and goals will always be the same, but the path taken to these objects can be infinitely different. The first movement taken by the robot during this calculation will always be of the same magnitude as part of its predetermined route. The movements direction will not be exact which is where the second movement and new heading will compensate. Using "(4)" and "(5)" in Fig. 16, the base calculations used in all the calculation functions is created. The difference between the functions is the way the calculations are applied. The calculations are applied based on how the robot should move after leaving a goal. This is due to how headings increase to turn clockwise and decrease to turn counterclockwise.

```
float phi, theta;
//Convert to radians to use in cos
perfectHeading1 = degreesToRad(perfectHeading1);
heading1 = degreesToRad(heading1);

phi = acos((pow(perfectSecond,2) - pow(resultant, 2)\
    - pow(first,2))/(-2 * first * resultant));

theta = heading1 + phi - perfectHeading1;

second = sqrt (pow(resultant,2) + pow(first,2)\
       - 2 * resultant * first * cos(theta));

beta = radToDegrees(acos((pow(resultant,2)\
     - pow(first , 2)\
     - pow(second,2))/(-2 * first * second)));

heading1 = radToDegrees(heading1);
```

Figure 16: Position Error Calculations

B.4 Ball Control

To control the balls within the robot, separate functions are created. These functions allow the robot to either score, eject, or intake a ball depending on what the robot sees in the conveyor assembly. Three distance sensors in the conveyor read if there is a ball in a certain slot. There are three slots in the conveyor, A, B, and C. The highest slot is slot A, the lowest slot is C, and in between is slot B. These distance sensors and an optical sensor read what the current layout of balls are in the robot and return a specific value. Other functions that adjust the ball's position before a certain task are coined by the same name with "Adjust" in the middle. These tasks are used before arriving to a goal or when approaching a ball. These optimize the placement of the balls for the next task. This minimized the amount of time the robot is still, thus decreasing the routine time.

Fig. 17 displays one of the many conditions used to detect the balls condition. Enumerators are used to make the program user friendly. In this example, "none" represents a value of 0. Other enumerators such as "want", and "nowant" are used as well with their values being 1, and 6, respectively. These values are assigned by another condition that reads the sensor values. The conditions that return the state use these values because a sensor could return completely different signals from when the robot runs through one condition to the other.

```
if((SLOTA != none)&&(SLOTB == none)&&(SLOTC == none))
{
  return 0;
}
```

Figure 17: Slot State Condition

Depending on the state of the balls in the conveyor, the robot will not do anything. This is a failsafe in the program. For instance, if the robot's desired routine were to approach a goal and place a ball at the top, the robot would place a ball in the goal if it had collected one. If the robot had failed at collecting the ball, it touches the goal and move on to the other goals. This maximizes what the robot could do in the allotted time.

V. CONCLUSION

The robot constructed consisted of a six 200 rotation per minute motors 15-inch by 13.5-inch drivetrain, two 600 rotation per minute motor intakes, two single-acting piston pivoting mounts, two 600 rotation per minute motor conveyor system, and a double acting piston hatch. Fig. 18 Displays the final robot design on Solidworks. Fig.19 shows the completed physical robot with all three subsystems. The robot also had line, infrared, distance, and optical sensors along with an inertial measurement units and encoders. A C++ program was constructed to have the robot navigate based on the user's pre-planned autonomous routine. The program uses functions to run the drivetrain around the field. Other side tasks are used to control the balls while the robot is moving. Classes are used to minimize the number of initialized variables when running the routine. Calculations and fail safes exist to reduce errors and to maximize the number of points scored in a routine.

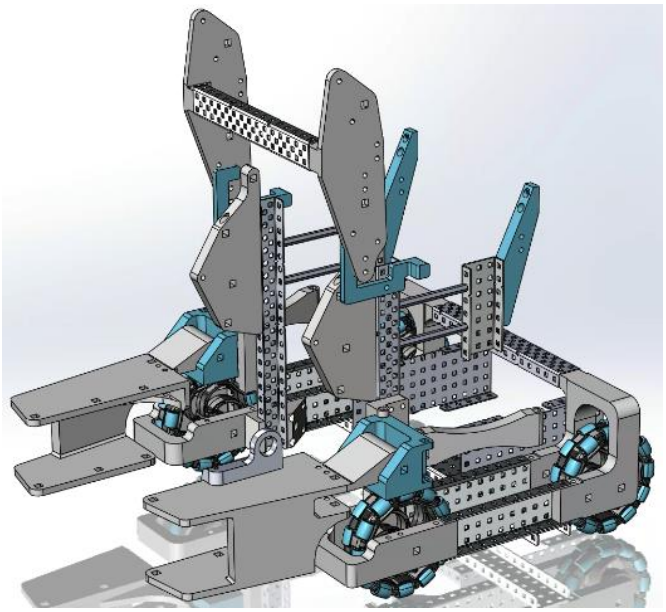

Figure 18: Final Robot Design CAD

Figure 19: Final Robot Design

VI. IMPLEMENTATION AND OUTCOME

The robot competed in an online event hosted by NASA engineers. The team ranked second overall and won the excellence award which is the highest award given to a team. The team is ranked fifth place in the world, with a high score of 205 points.

ACKNOWLEDGMENT

The author of the paper would like to thank the President of Vaughn College of Aeronautics and Technology, Dr. Sharon DeVivo. Special thanks are given to Dr. Hossein Rahemi, Department Chair of Engineering and Technology, along with professors Shouling He, Ryan Tang, and Donald Jimmo. Finally, thanks are given to lllDMAX for providing the filament for this project.

REFERENCES

[1] VEX Robotics, https://content.vexrobotics.com/docs/vrc-change-up/Appendix-B-12012020.pdf, Accessed March 2021

[2] VEX Robotics, https://www.vexrobotics.com/276-2154.html#ogkppei, Accessed March 2021

[3] VEX Robotics, https://www.vexrobotics.com/276-4855.html, Accessed March 2021

[4] VEX Robotics, https://www.vexrobotics.com/276-4852.html, Accessed March 2021

[5] VEX Robotics, https://www.vexrobotics.com/276-7043.html, Accessed March 2021

[6] THORLABS, https://www.thorlabs.com/newgrouppage9.cfm?objectgroup_id=9013, Accessed June 2021