Engineering portfolio for FTC 2022/2023

Team: Montenegro Robotics

Team No. 22754

Table of Contents

Overview

Introduction

In this engineering portfolio we will be discussing the design decisions and design process that we had to implement during the making of our robot for this year's First tech challenge "Power Play". Aside from the realized implementation we will be discussing ways to improve upon the design.

Design overview

When discussing the design of our robot we can separate it in 2 parts:

- Drive platform
- Mechanism for moving the game elements.

These 2 parts need to work together ensure optimal performance.

The biggest challenge was the limited volume of the robot as well as its need to place game elements on high poles. For our solution we implemented a drive platform that utilizes 4 mechanicum wheels to enable omnidirectional movement. As well as a 4 DoF[1] robot arm that can fit within the size limit (45,73cm x 45,73cm x 45,73cm) while still being able to score on the highest pole, see Figure 1. Unlike other solutions it allows not only for vertical movement but also horizontal movement. This large working area combined with its ability to change the attack angle when picking up a cone makes it an optimal solution for picking up both up straight cones as well as ones that fell over.

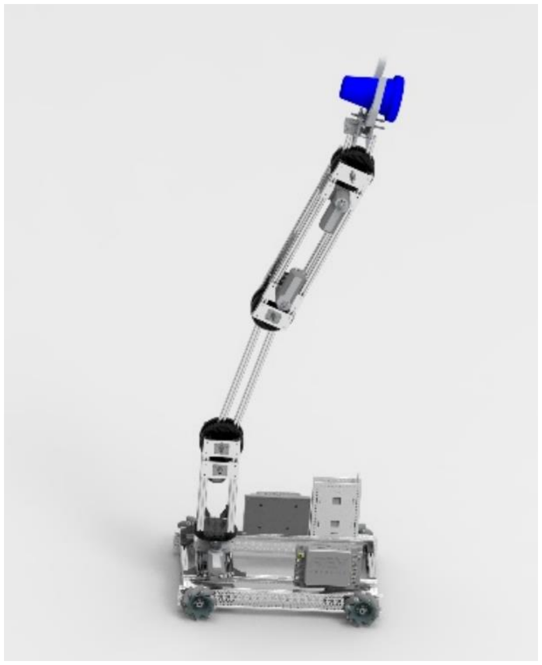
Figure 1 Computer rendered CAD model of the robot

[1] Degrees Of Freedom

Mechanicum drive platform

For the construction of the drive platform 4 mechanicum wheels were used. To enable omnidirectional movement each wheel needs a separate motor. Many factors led us to decide to use hex core motors. Their high torque and small dimensions were ideal for our use case.

Construction

These wheels are mounted together with the motors on 2 parallel aluminum extrusions. Two smaller aluminum extrusions connect them and are placed normal to the first 2. Together they form a rectangular frame. This frame is the base of the robot and thus it needs to be stable. To achieve this the wheels are mounted as far away as possible from each other.

Mathematics of movement

Mechanicum wheels are produced in such a way so that they produce a pulling force that is 45 degrees of axes of rotation and perpendicular to the ground, see Figure 2.

The total pulling force is a vector sum of all the individual pulling forces. Adjusting the power of the motors allows for omnidirectional movement.

$$\vec{F} = \vec{F_1} + \vec{F_2} + \vec{F_3} + \vec{F_4}$$

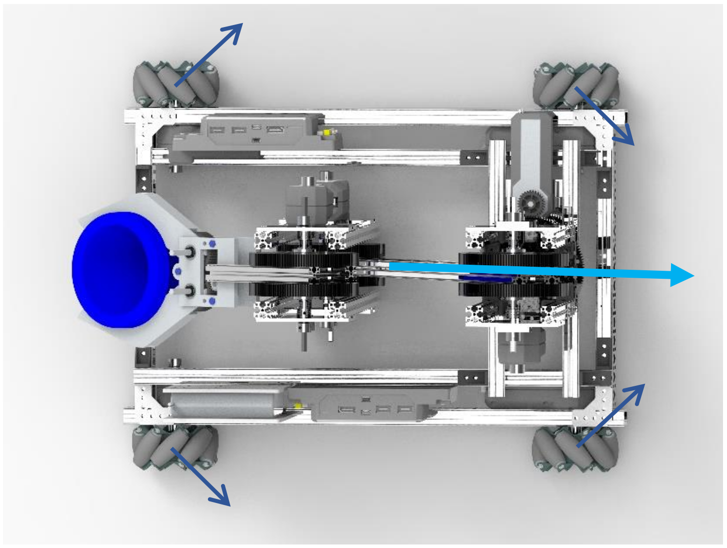

Figure 2 Mechanicum wheels torque

Robot arm

Design overview

Due to the limited number of motors the arm has 4 DoF. The approximate working area of the robot arm is a sphere with a radius of 1m. The arm is equipped with a hand designed to pick up cones. This hand is powered by a servo motor while the rest of the arm joints are powered by a hex core motor with an additional 15:125 gear ratio. This allows for precise movement and high torque while almost eliminating the backlash present in the motors.

The arms 4th DoF allows for full 360-degree rotation of the whole arm. It is powered by a hex core motor with a 30:125 gear reduction.

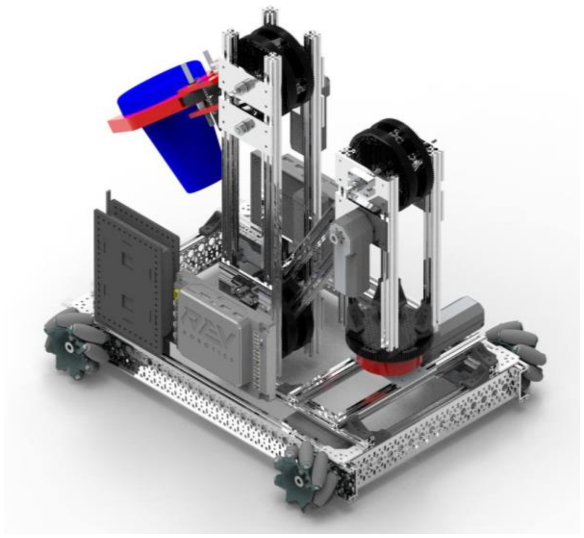
Figure 3 Computer rendered image of the robot

Parts selection

The main goal of this arm was precise movement with little to no backlash. Additionally, we needed to keep it low mass with high stability. After many revisions of the design, we settled for aluminum extrusions connected with costume made aluminum parts, see Figure 4.

For the rotation axes of the whole arm a bigger bearing was needed. A custom 3d printed holder was made with screws as reinforcements for high loads.

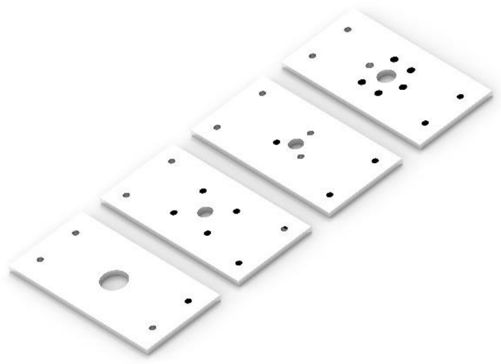
Figure 4 Computer generated custom aluminum parts

Construction

This arm is constructed from 4 individual segments. These segments are made mostly from aluminum.

Second segment is just 2 aluminum extrusions that connect segment 1 and 3.

Third segment has 2 motors that drive the third and fourth joints. If is similar in construction to the first segment. Forth segment consists of the 2 small aluminum extrusions that connect the grabber to the joint.

The grabber is 3d printed from 4 parts incorporating preexisting gears to improve reliability and reduce wear. The inside has rubber coating, see Figure 5.

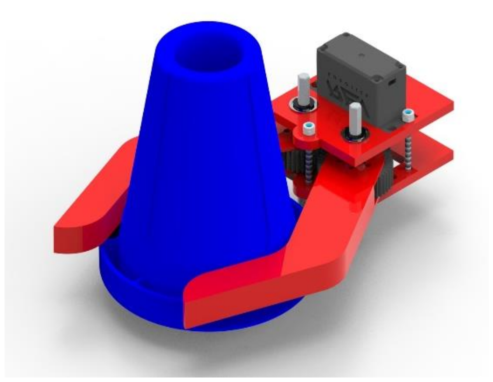

Figure 5 Grabber mechanism

Base bearing

The 4th DoF is a rotation of the whole robotic arm mechanism. For this purpose, we have used a 3205 ball bearing to support axial as well as radial loads in a combination with a 125 tooth gear paired with a 30 tooth gear mounted on a DC Hex Core Motor. To hold the bearing and allow for mounting we have incorporated 3d printing technology to manufacture the mounting system. This mounting system consists of inner and outer ring made from PLA plastic and 12 M4 screws that act as reinforcement as well as allowing for easy mounting, see Figure 6.

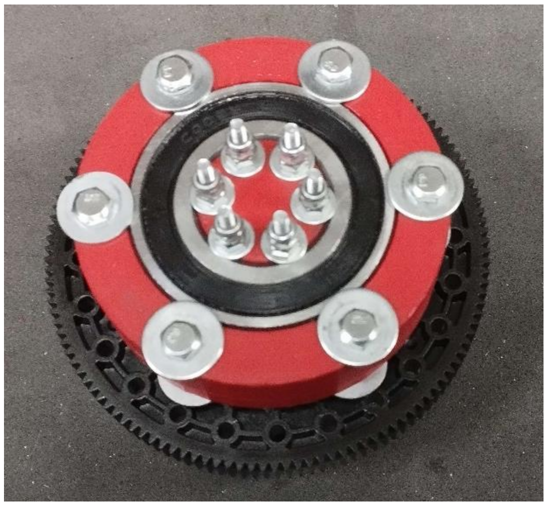

Figure 6 Bearing mount mechanism

The 3205 ball bearing is a Double row angular contact ball bearing, see Figure 7. This was the main reason for picking it as it provided great performance without the need for the use of Slewing bearings.

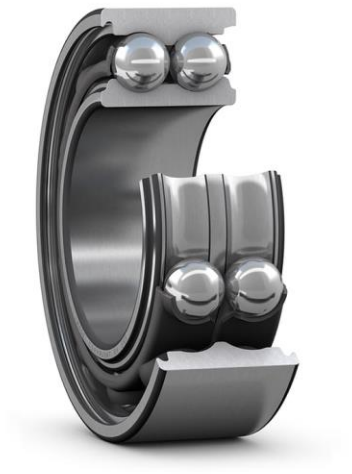
Figure 7 3205 Ball bearing (SKF, n.d.)

Invers kinematics

When controlling the robot arm it would be a near impossible task to control the individual joints to move the arm to a position in 3d space. This is why we implemented inverse kinematics. They allow us to define a position and calculate the angles needed to get to that position. Using inverse kinematics, we can make an algorithm to automatically move the arm to a defined position.

To derive the formula for the inverse kinematics we look at the last 3 segments of the arm as 3 vectors. The final position is described as a vector sum of the vectors. We want to define the position and attack angle when picking up the cone. Aside from these inputs we have constants for the length of individual segments. We derive the formulas needed in the following way:

Inputs: $X_0, Y_0, \gamma, a, b, c$

$\gamma \rightarrow$ Attack angle of the arm

$$X_1 = X_0 - \cos\gamma \cdot c$$

$$Y_1 = Y_0 - \sin\gamma \cdot c$$

$$d = \sqrt{X_1^2 + Y_1^2}$$

$$\varphi = \tan^{-1}\left(\frac{Y_1}{X_1}\right)$$

$$\phi = \tan^{-1}\left(\frac{\frac{a^2 - b^2 + d^2}{2d}}{a}\right)$$

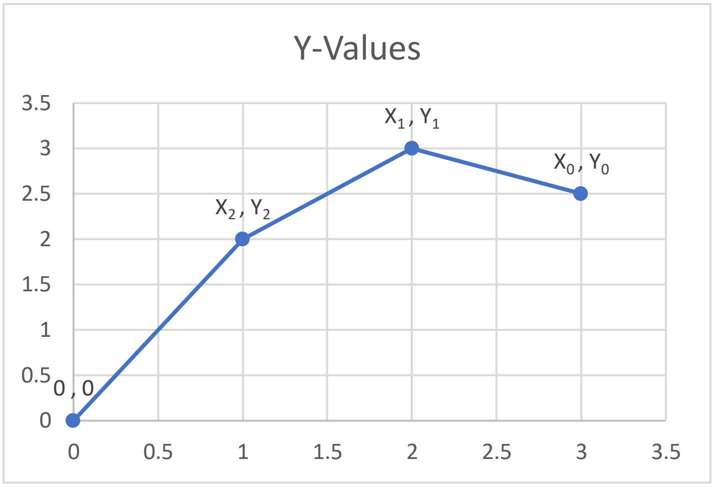

Figure 8 The vector representation of the robot arm

$$\alpha = \varphi \pm \phi \rightarrow \textit{There are 2 solitions for this inverse kinematics}$$

$$X_2 = \cos\alpha \cdot a$$

$$Y_2 = \sin\alpha \cdot a$$

$$\beta = \tan^{-1}\left(\frac{Y_1 - Y_2}{X_1 - X_2}\right)$$

We have ignored the rotational axes of the whole arm as we have decided that the added complexity of 3d inverse kinematics was not necessary as defining the position with an angle of the arm and X, Y coordinate was enough.

Angle correction

From these formulas we get an angle between ground and the segment. We need to convert this to encoder steps. The steps that we take to do this are:

- Fallback to last valid angle in case NaN is detected. NaN is present when the arm goes outside the working area.
- Correct the angle transition from atan2() function. It returns a value between π and -π or 180 degrees and -180 degrees respectably. In case of a transition from 180 degrees

to -179 degrees (1 degree change in angle) the arm would do 359 degrees change in angle. This is the reason for correcting the angle to be continuous. To achieve this, we compare the last valid value with the different corrected value and using the one that results in the smallest angle change.

- Converting an angle between the segment and ground to an angle between previous and next segment.
- After this we convert the angle to steps with a constant determined by the gear ratio.

PID control

Overview

PID controller is a commonly used type of controller that allows for better tuning of a closed loop feedback system. Its use is not limited to just motor position, it can also be used for velocity control. In our code we implemented a custom PID controller for the purposes of autonomous movement to a precise position.

Working principle

The basic functioning principle of the PID controller is that the power is equivalent to the Proportional, Integral and Derivative of the error times an appropriate constant. This is the reason for the name PID. The working principle can also be expressed in a formula.

$$Power = error \cdot kp + \int error \cdot dt \cdot ki + \frac{derror}{dt} \cdot kd$$

Implementation

We only use the PID controller during an autonomous period. We found that the built-in function DRIVE_USING_ENCODER was good enough for manual movement where real time corrections can be made by the driver.

PID controller allows us to make autonomous period more reliable without adding too much complexity to the code.

Driving enhancements

Overview

One of our highest priorities was the driver experience. We didn’t want for the process of scoring a cone to be hard or complicated. This is the reason why we implemented many driver enhancements. The purpose of a driver enhancement is to make the job easier, safer, faster and more consistent. We implemented the following enhancements.

Analog control for the speed.

Mechanicum wheels can be unintuitive to control individually to achieve proper pull force direction and intensity. This is why we implement an easier way to do so. It takes a single x y coordinate of the right stick on the controller and converts it to power and angle of the resulting force. It than sets the motors to appropriate power levels. It also allows for rotation using the second analog stick. The math used for that is as follows:

$$Power = \sqrt{x^2 + y^2}$$

$angle = \tan^{-1}\left(\frac{y}{x}\right) + 45°$ The function Math.atan2(y, x) would be used to get the scope of values from 180 degrees to – 180 degrees (π to -π in radians).

Power for the first wheel pair

$$Power1 = \sin angle \cdot Power$$

Power for the second wheel pair

$$Power2 = \cos angle \cdot Power$$

The arm also has analog control as its x and y coordinates as well as the attack angle for picking up the cones. The change in angle/coordinates is proportional to the intensity of the analog stick.

Preset coordinates for the driver period

We have also made some preset coordinates for easier arm control. The preset coordinates are divided into pick up positions and scoring positions. They are designated on individual buttons on controller. To prevent rapid movements of the arm that might nock the robot down we have implemented interpolation.

The out of bounds positions

The arm has been programed to keep track of its last valid position. This helps in combating not a number error when going out of bounds. With this feature we ensure that the robot does not go back to the default position unintentionally.

Autonomous period

For the autonomous period we have used the PID controller to ensure more stable and consistent results. We have found that this significantly improves the autonomous period. Because the complex movement needed to score the cone with the arm we found a good balance between the speed and precision. It allowed us to score 2 cones on the high pole and have enough time left to finish parking. Additionally because of the PID controller every part of the robot keeps its position and moves back to it if moved unexpectedly. This allows the scoring even after the collision with a different robot during the autonomous period.

Signal sleeve and it’s recognition

The sleeve has been designed with simplicity in mind and so it consists of 3 colors one for each image. This allows us to use a color sensor instead of a camera. Because we are using a color sensor we are taking an average of the RGB values over a short period of time to reduce error and give more consistent results. This is done by adding up values and dividing them by the number of samples taken.

Possible improvements

Overview

We worked really hard on making our robot as best as possible but there are still many things than can be improved. Despite this fact we are happy with the current state of our robot and hope to improve in the future while learning additional skills along the way.

Encoders

The encoders present in all the motors we used, aside from servo motor, are 4 count per rotation incremental encoders. This limits the accuracy of the position as well as providing only a relative change of position rather than a absolute angle.

This problem can be fixed in many ways some of witch include:

1. Potentiometer for analog feedback of the current position. Drawback of this method is the maximum

operating speed of classical potentiometers as well as wear over time. These issues can be addressed by the following methods.

2. Magnetic absolute encoders can be used to get the absolute angle of the individual joints for further use by the program to accurately calculate inverse kinematics. External magnetic fields may affect the noise and accuracy of the output.
3. Optical absolute encoder solve the issues of the previous 2 but come at a significantly larger price point.

Limit switches

Adding limit switches is a very good practice for any machine that is capable of self-intersecting. These limit switches can be a hardware feature but in a lot of cases can be programed in on the software side. Hardware switches have a close to 0% error rate compared to software side implementation that relays on forward kinematics to calculate are intersections present.

Spring counterbalancing

The encoders present in the motors that drive the arm can provide a closed loop feedback system that allows for the robotic arm to keep its position. This comes at a drawback as the motors almost always need to counteract a constant gravitational force. This reduces efficiency and max reliable operating speed. This also make the system of the robot arm a nonlinear system thus we can’t have a perfect PID controller for all the movements.

A solution we came up with but didn’t implement was spring based counterbalancing. This can increase the linearity of the system as well as increasing the overall efficiency at the cost of simplicity. This would allow for easier movement of the arm. Because of its complexity we decided in the end we decided not to implement it, but it was an option we were considering during the design of our robot.

Machine vision

The addition of machine vision could drastically improve our autonomous period. The only reason we didn’t implement in in our robot is the lack of skill that we hope to develop in a near future.

Awareness of the position in 3d space

Adding additional sensors to determine the location and orientation of the robot within the field can allow for the robot to correct its position in real time and improve its autonomous period.

Conclusion

We learned a lot of important skills during the making of our robot. WE hope to be able to improve our skills even more as well as inspire more people to get interested in the hobby of robotics and programming.