17089

2022/2023

Engineering Portfolio

POWERED BY Redstone

Open Gate School

#17089

TABLE OF CONTENTS

1. INTRODUCTION

1.1. TEAM MEMBERS

PETR KUŽELA

Since my childhood, I have enjoyed building and tinkering with LEGO. The creative and constructive skills have nicely transferred into designing robots for FTC. I hope to continue my creative development into my future profession.

OLIVER ŠKAŘUPA

Working part mostly in engineering, but doing some coding as well. I am proud that we used our experience from previous seasons to improve upon our workflow and learn from our mistakes. I am the biggest fan of the CAD program OnShape due to its collaborative features. My favorite thing about Robotics is the necessity for teamwork and collaboration that leads to unique solutions for every team.

DANIEL STRNAD

I was a weird child… I've always preferred to build Lego exactly according to the manual. And that endured until I was able to make my own Lego bricks.. Thanks to robotics. It simply satisfies my engineer heart to model a part in OnShape and then see it being materialized into a real thing.

MARTIN LIBOR MATURA

Ever since I was a kid, I have always been fascinated with technology. That is why I immediately jumped at the opportunity to join the robotics team, so that I can pursue and express my passion.

ROSTISLAV JANOVSKÝ

This is my second year in Powered by Redstone. I really enjoy learning new stuff, even though it is sometimes really hard. It is amazing that we have a chance to work on our own robot, so I wanted to be a part of it. The best thing about robotics is the freedom of designing and building.

TOMÁŠ JIRMANN

From a very young age, I knew that I wanted to be an engineer. Starting from my fascination with LEGO bricks as a small child and continuing with my passion for designing in OnShape today. I am very grateful to be a part of the robotics team because I feel like it gives a higher purpose to engineering as my hobby.

POWERED BY REDSTONE

Team *Powered by Redstone* is a Czech FTC team that consists of members studying at Open Gate School in grade ten, eleven and twelve. Open Gate has been mainly a language school in the past, but it has decided to promote STEM fields during the recent years. Our robotics team serves as one of the ways to achieve this goal. As a part of this project, our team is extensively supported and funded by our school and we were given our own academic club and a full workshop.

Open Gate is a fairly small school, as it has only about three hundred students. Even so, we have 6 active members with occasional help from others. All of our members are passionate about robotics in general, and many of us would like to pursue the goal of working in the field of engineering. As a result, even though our school schedules are not ideal, we are still meeting at least once a week in the late afternoon to actively work on the robot. Furthermore, our members are willing to sacrifice their free time in order to work on our robot during the week or even during weekends. We are taking part in FTC for the fourth year with previously participating in two tournaments hosted in Barcelona.

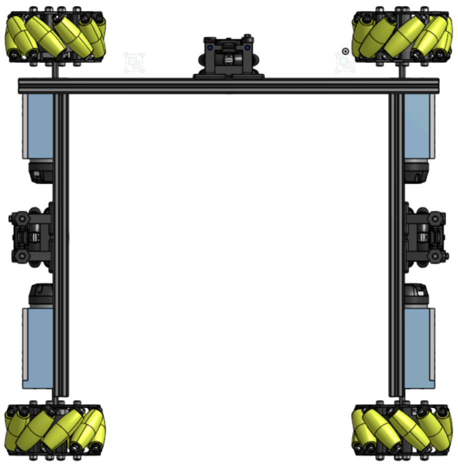
2. ENGINEERING

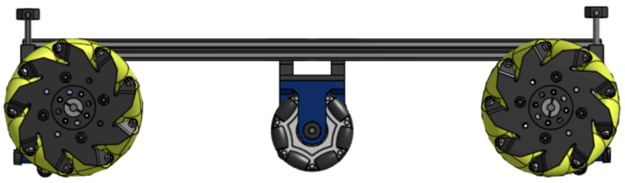
2.1. INTRO

Last year was full of challenges, including team management difficulties and a general lack of a concise vision for the robot. For the Power Play season, we decided to build the robot around a specific vision: to build the most modular robot, which can be upgraded, modified and have its parts replaced in minutes.

2.2. MOTOR MOUNTS - DRIVETRAIN

Inspired by last year's challenge, where the preferred strategy was to make the chassis of the robot thinner, we decided to bring the vision of modularity to the chassis itself. To properly achieve this, we had to think outside of the box (within robot boundaries, of course). Whereas most chassis of robots have their mecanum wheels oriented like a car might, with the faces of the wheels facing to the sides, we embraced the true nature of the omni-directional wheel and mounted the wheels facing the front and rear. While keeping the full range of motion, this allowed us to keep the motors and the side odometry wheels mostly within the profile of the mecanum wheel, freeing up a lot of space in the middle of the robot.



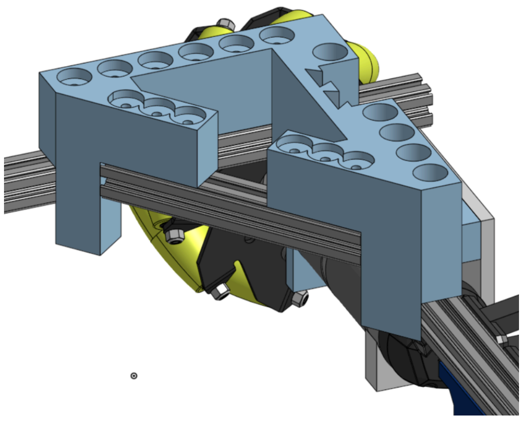


Every component on each side is mounted on a single REV robotics metal extrusion, with another extrusion connecting them. This allows us to change the dimensions of the robot with ease, only by sliding individual parts of the chassis within said extrusions. The mecanum wheels are mounted directly onto motors, which we found suitable for our design, as gearboxes would unnecessarily hinder the modularity of the chassis. To ensure that the load onto the motors is properly transferred to the chassis, we use custom brackets to connect the motor to the extrusion in multiple places.

An example of the modularity of this design is that when we found the spaces between the junctions to be very tight for drivers, we shrank the robot by multiple centimeters in width, so that we can navigate the field more easily,

2.3. CHASSIS CONNECTION

To take full advantage of the space unlocked by our motor layout, we decided not to connect the robot at both front and rear, as the space inside the robot could be used to handle game elements. Therefore, we had to come up with a way to connect the chassis at the rear. The solution we adopted was to 3D print large plastic brackets, which we would reinforce using metal extrusions. It was also essential that the part did not allow the chassis to flex during operation. To ensure the security of the connection, we used an excessive amount of bolts, which were fastened under tension (the chassis was forced together at the front). This allowed us to decrease frontal oscillations from 3 mm to 1 mm, which we find acceptable as a tradeoff for the increase in manipulation space. The

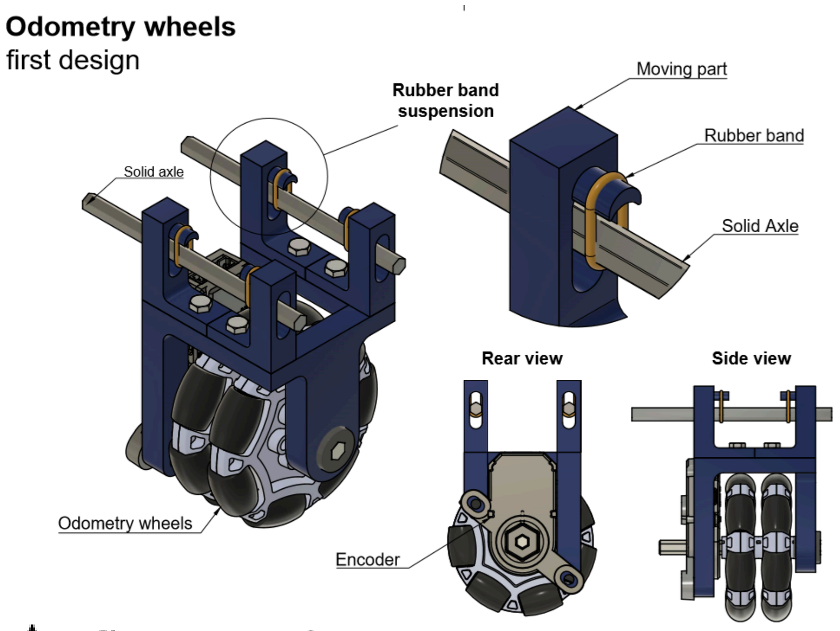


extrusions in the brackets also allow for mounting of electronics and other functional parts of the robot.

A downside that we found to having the robot be U-Shaped is the tendency to mount functional elements and electronics at the rear. This leads to an unbalance in load distribution between front and rear wheels, causing driving difficulties. The solution is simple, as the modular nature of the robot allows elements of the robot to be re-positioned on the chassis. Moving the electronics to the front to offset other functional elements fixed all our balance issues.

2.4. ODOMETRY MOUNTS

In order to know the precise position of the robot on the playing field, we needed to implement odometry encoders. The solution was placing three encoders reading data from their independent wheels.
To increase data accuracy, we created a custom 3D printed odometry design with a separate suspension system. Each encoder is set up to be pushed into the ground by rubber bands in order to maintain constant contact with the ground, even when driving over irregular terrain. Our first design is outlined in *figure 2.6/a*. Two axles are mounted to the robot's chassis using a simple part. The odometry encapsule is then slid on the two axles and attached using rubber bands. The rubber bands pressure the odometry wheels into the ground and therefore maintain contact with the ground even when driving over irregular terrain. With this solution, the code can fully rely on position data from our odometry encoders, and use that for advanced control of the robot's movement.



↰ figure 2.6./a

However, this solution did not work as well as we had hoped. There were many problems. The first big issue was with the odometry wheels moving sideways because the hole for the axles was too big. If we made the hole smaller, the axles wouldn't be able to slide easily. This turned out to be a major problem for this design.

After coming up with some new ideas, we decided that the best solution is a redesign of the odometry suspension. The new design uses only the part that attaches to the encoder and holds the wheels. This design has sliding axles attached to the wheels positioned vertically. The two axles slide in a rig attached to the main chassis of the robot. This solves our problems because it allows the odometry setup to only move vertically with almost no side movement.

Odometry wheels
final design

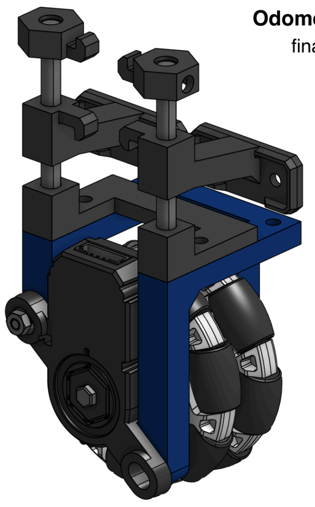
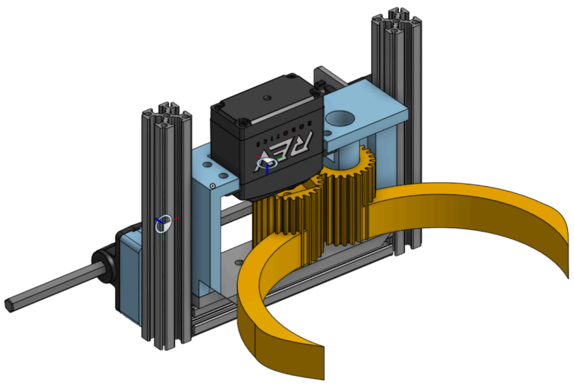


figure 2.6/c

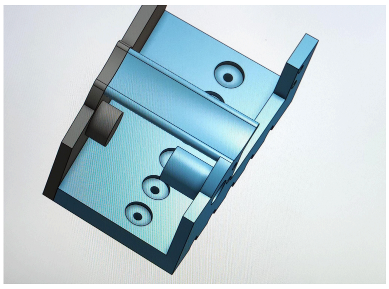
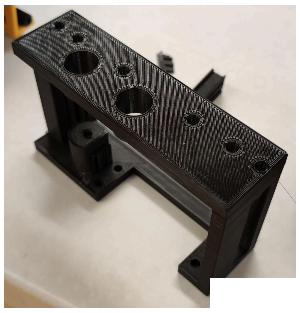
Comparison of designs	First design with horizontal axles	Final design with vertical axles
Positives	• Low profile- allows for parts above it • Rubber bands can be attached easily	• No travel in any directions except for vertical • Adjustable pressure on wheels • Easier to assemble
Negatives	• Some side movement • Can move freely on the Y axis of the robot.	• Rubber bands must be tied around hole

2.5. END EFFECTOR - GRABBER

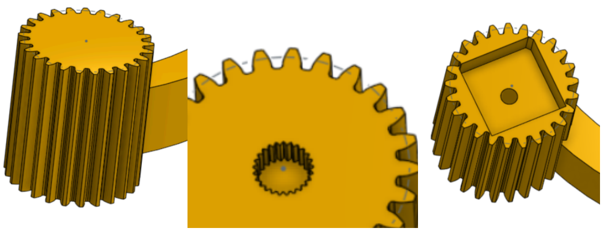


The grabber assembly began with finding a mechanism to hold the cones using one smart robot servo. We developed two clamps attached to gears - so the clamps would move specularly.

With the first prototype of clamps, we started to develop the part to hold them together and to attach the mechanism to the rest of the robot.
We modified the attachment mechanism. The first prototype was very bulky, so we have decided to rework the whole thing to be as light as possible. Figure below shows the first and final prototypes of our holder

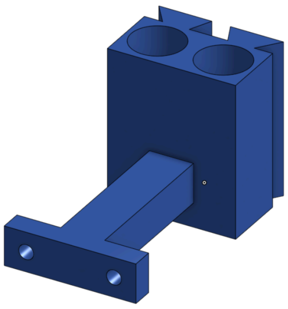
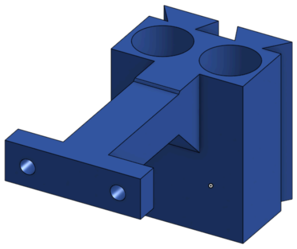
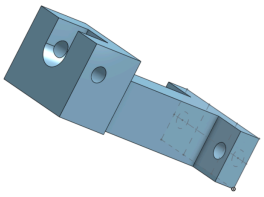



However after first testing the part, it had one critical flaw. The plastic extrusion fitted on Smart servo shown on Figure 2.5.b was abrased by the torque produced by the servo. We decided to solve this by using a more sturdy aluminum piece on the servo gear and thus increase the radius from the axis of rotation and avoid application of pressure on fine plastic faces. Figure above shows the development of servo-clamp attachment in CAD. The grip of the clamps was solved by application of silicone.



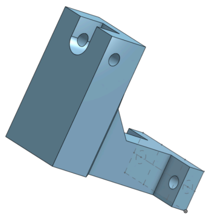
2.8. SIDE PLATES

When the side plates were designed, we had to invent a mechanism to attach them to our chassis. We aimed to create an easily removable mechanism, while fulfilling the modular complexion.

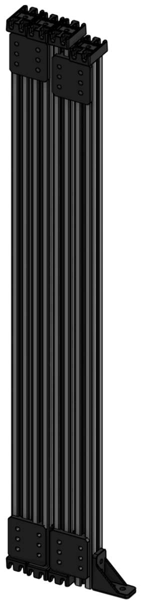




Firstly, we solved the attachment at the rear by inserting it into the structural bracket. The mount itself therefore serves a structural role in the robot. For the front holders, we used an extrusion in the back side of the side plate with corresponding insertion point on the H shaped side plate mount.
When everything was ready for printing, we found out that our halves of side plates were still too large to fit on our 3D printer. This challenge has been solved by changing the angle of cut and the sideplate assembly could finally have been printed. However, that was when new challenges arose. Due to material bend, which was not taken into account in the CAD model, the side plates were positioned too low to make the robot operable. Thanks to the clever design of both side plate

holders, the vertical position of side plates can be easily manipulated, so we are able to change between designs of holders for two possible heights within minutes. The same thing would be possible with X positioning thanks to the highly modular nature of side plate holders while keeping the side plates themselves the same. Figure on the left shows the modularity along z-axis of back side plate holder (dark blue) and front side plate holder (light blue)



2.9 Linear Manipulator (Scrapped)



Our first idea for creating a mechanism for lifting cones on the poles was to create a linear lift. This lift would be created with REV robotics parts and operated by winding string around a wheel. We wanted to have one lift on each side of the robot to improve stability and a grabber mechanism mounted on top that would hold the cones during the lifting movement. However, when we created this concept in reality, the lift had too much friction, and could not be lifted smoothly. After trying some fixes to reduce friction, we ended up deciding on completely scrapping the design and trying something different.

Our next idea was to utilize sliding ball bearings in between REV extrusions that would reduce friction. However, when we created a prototype for this sliding motion, the ball bearings were not contained securely. In the end, we decided to scrap linear motion and change our plans.

2.10 New Manipulator

The new manipulator design consisted of an arm lifted by a chain driven mechanism that allows the cone to be lifted to the desired height in one movement. Attaching the static part of the arm securely to the chassis of the robot was a difficult problem to tackle. To reduce any unwanted movement in the arm, we designed and 3D printed a custom extremely rigid mounting part that secured the extrusions together with an abundant amount of screws. This ensured that the lift and chassis are connected tightly and completely eliminated any movement in the pivot of the arm.
Another challenge was driving the arm. We chose to use chains and gears to create a desired gear ratio. However, the high amount of torque induced on the plastic gear

created issues. To fix this, we needed to modify metal REV High Strength Hubs to reinforce the gears.

3. PROGRAMMING

3.1. INTRO

If you take engineering as the body of our robot, then by that analogy, programming would be the brain. The code makes the robot act. Starting from simple actions, like driving around the field and lifting cones, to more complicated ones like calculating trajectories and analyzing camera images. This year we decided to tackle a big task of creating our own odometry algorithm. With this algorithm, our autonomous mode will become way more precise and it will also simplify many actions for our drivers in TeleOp. Our mentor Daniel Lessner also plays a crucial part in our team. With his PhD in computer science, he provides valuable insight and makes our lessons very educational.

3.2. SET-UP

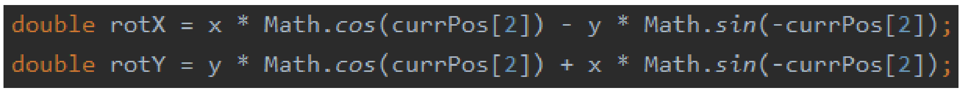
From the beginning of our team we always used Android Studio for developing our robot. We also decided to use Android Studio because we wanted to have more flexibility with the repository as this year we created our own image recognition algorithm for detecting the cone sleeve position using OpenCV.

Furthermore, we use the linear op mode classes to develop our op modes. The process began with forking the new repository provided by First on Github and setting it as our own repository we can all collaborate on through Git and Github. Furthermore, we use 2 Motorola G5 phones that communicate through Wi-Fi direct to control our robot.

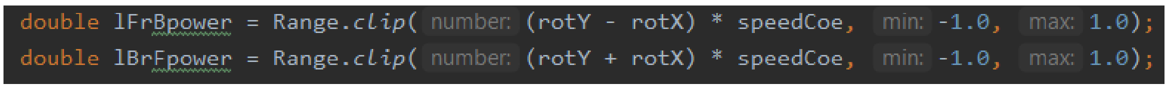
3.3. TELE OP

At the beginning of the year, we made the commitment that the most important part of the tele op mode to us is driving. This is for one reflected in the effort that went into our chassis. Furthermore, the first thing we focused on as programmers was the mecanum wheels gamepad control algorithm. The basis is a field-relative mecanum drive algorithm.

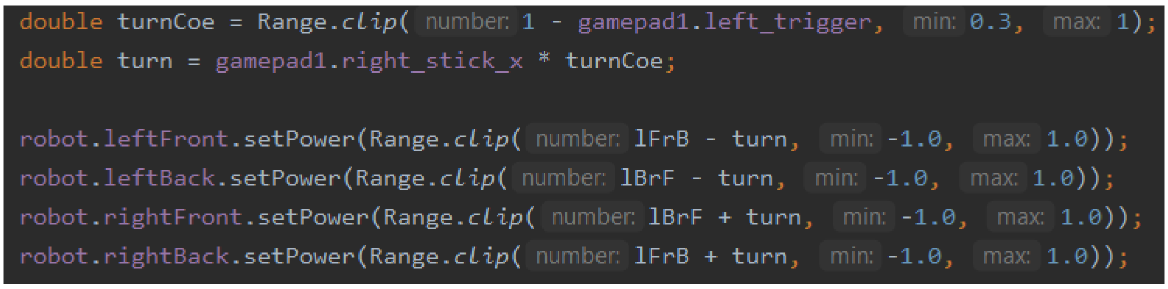
Because our robot is equipped with 3 dead odometry wheels (more on them later), we can easily and precisely track the position and more importantly rotation of the robot at any time. Thanks to this, creating the field-relative drive algorithm was not a difficult task at all. *currPos* is an array of 3 double values that store the robot's x, y and rotational position and therefore, *currPos[2]* gives the angle at which the robot is currently rotated from its starting position. *turnCoe* and *speedCoe* are coefficients that we introduced into the code to boost or downplay some inputs and thus allow for both higher precision and higher velocities when needed.



```
double rotX = x * Math.cos(currPos[2]) - y * Math.sin(-currPos[2]);
double rotY = y * Math.cos(currPos[2]) + x * Math.sin(-currPos[2]);
```



```
double lFrBpower = Range.clip( number: (rotY - rotX) * speedCoe,  min: -1.0,  max: 1.0);
double lBrFpower = Range.clip( number: (rotY + rotX) * speedCoe,  min: -1.0,  max: 1.0);
```



```
double turnCoe = Range.clip( number: 1 - gamepad1.left_trigger,  min: 0.3,  max: 1);
double turn = gamepad1.right_stick_x * turnCoe;

robot.leftFront.setPower(Range.clip( number: lFrB - turn,  min: -1.0,  max: 1.0));
robot.leftBack.setPower(Range.clip( number: lBrF - turn,  min: -1.0,  max: 1.0));
robot.rightFront.setPower(Range.clip( number: lBrF + turn,  min: -1.0,  max: 1.0));
robot.rightBack.setPower(Range.clip( number: lFrB + turn,  min: -1.0,  max: 1.0));
```

3.4. CONE SLEEVE DETECTION

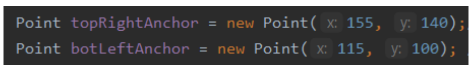
A crucial part of the autonomous period in this season is being able to correctly detect the orientation of the cone sleeve, and then park the robot in the according area. Performing this task correctly with a custom created signal sleeve will yield 20 points, which is why we knew from the beginning that this is something we want to focus on, so that we can score these 20 points consistently in competitions.

In the previous seasons, we used an outdated and very limited camera OpMode library to access the outputs from the RC phone camera directly and then analyze the images. This year, we changed our approach and decided to learn to use OpenCV for camera detection. The learning process came with its challenges at first, however, through practice we soon became proficient in using the library with all its features that were relevant for us.

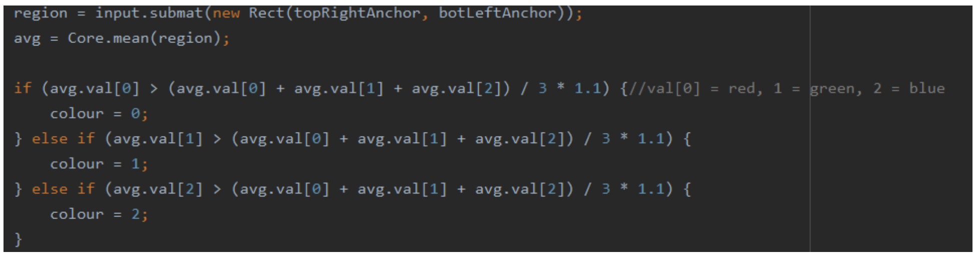
Once we were capable of using OpenCV to our advantage, it came time to think about the algorithm that we want to use to detect the orientation of the cone. OpenCV is not the best tool for image recognition, but it is great at image processing, which is why we decided to design our custom sleeve so that each side would mostly consist of one color, and as such, after getting the average color of the pixels when scanned, the side of the cone sleeve could be easily distinguished. After brainstorming ideas, we landed on the decision that the algorithm would first locate the cone in the image by searching for a consistent row of red pixels. Once it found a row of red pixels that would satisfy some preset conditions to be considered the base of a cone (mainly length), the entire cone would be outlined by a rectangle formed based on the base line of red pixels. Then, the only remaining step is getting the average color inside the rectangle and checking which side of the cone sleeve the average color is closest to.

When it came to implementing this algorithm, the part of detecting the base of the cone proved to be more challenging than we expected. We unfortunately could not find a way of consistently detecting the base row, as the algorithm kept getting distracted in the gray field floor. Even after trying to convert the image into every possible color space and trying to detect the red line through looking at every possible parameter, we were unable to create a version of the algorithm that was consistent enough. Even after trying different approaches such as measuring changes in red pixel values across rows or checking numerous rows above each other, we still couldn't produce a result that we were satisfied with, not to mention that the pipelines took way too long to execute anyways.

For this reason, we finally made the decision to hard-code the position of the cone within the camera image and commit to relying on precisely positioning the phone in the same way every time. Somewhat interestingly, this process proved to be much more reliable than any of our previous attempts at detecting the base of the cone, and so we stuck with it. Now, all that the algorithm does is take the average rgb values within the predefined rectangle and check whether the resulting average is the most red, green or blue. Below is a screenshot of the main part of this algorithm:



```
Point topRightAnchor = new Point( x: 155,  y: 140);
Point botLeftAnchor = new Point( x: 115,  y: 100);
```



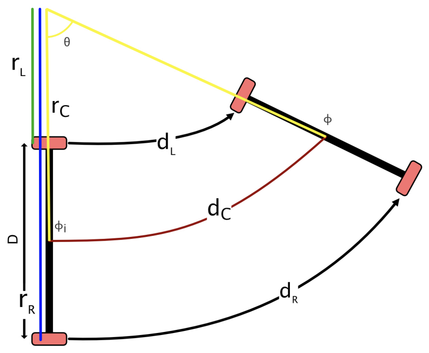
```
region = input.submat(new Rect(topRightAnchor, botLeftAnchor));
avg = Core.mean(region);

if (avg.val[0] > (avg.val[0] + avg.val[1] + avg.val[2]) / 3 * 1.1) {//val[0] = red, 1 = green, 2 = blue
    colour = 0;
} else if (avg.val[1] > (avg.val[0] + avg.val[1] + avg.val[2]) / 3 * 1.1) {
    colour = 1;
} else if (avg.val[2] > (avg.val[0] + avg.val[1] + avg.val[2]) / 3 * 1.1) {
    colour = 2;
}
```

3.5. ENCODERS & ODOMETRY

In order to track the position of our robot on the field, it is equipped with 3 dead odometry wheels that are connected to REV Through Bore encoders. We created our own odometry algorithms for tracking the robot's position and moving target points, which we will describe below.

To calculate the displacement of the wheel that the encoder is attached to, it is important to know the diameter (d) of the wheel, for the circumference (C) to be calculated. $C = d\pi$ Furthermore, as in the previous example, I will assume that the encoder disk has 360 contacts so that one rotation translates to a number of 360. The current number of contacts made will be represented by the variable a. Therefore, the displacement traveled (dt) by the wheel can be calculated by using the equation of: $\frac{a}{360} * C = dt$ The variable is divided by 360 to get the fraction of how much the disk has been rotationally displaced. Later, when multiplied by the circumference, the variable (dt) for displacement is obtained.

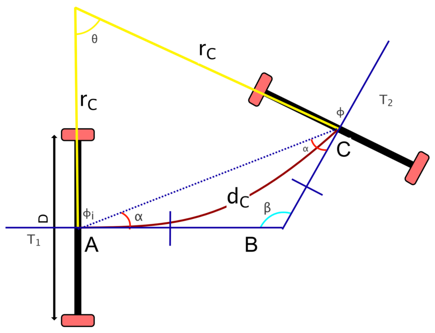
For the parallel wheel tracking, arcs can be used to calculate the change in displacement. It is important to mention that even driving in a straight line can be considered a movement in an arc with a radius of infinity.



The figure demonstrates an example of how wheel displacement can be expressed through an arc of length. The computer that will be doing the calculations, will use the values from the encoders every amount of milliseconds (depends on the speed of the computer). Therefore, the algorithm will be calculating the change in displacement between two instances in time or also called “frames”.

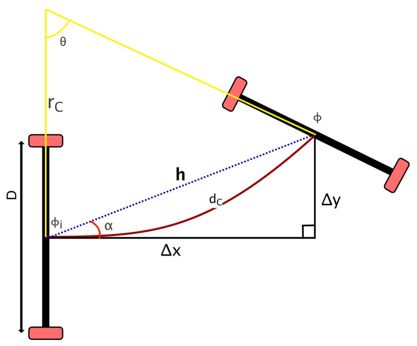
$d_C = \frac{d_r + d_L}{2}$ The central displacement dC is equal to the average of dR and dL.

The equation for theta works out to be: $\theta = \frac{d_R - d_L}{D}$ Theta is the angle the robot turns during the frame.



There are some relations that should be pointed out. Firstly, a quadrilateral is made of the two radii and the two tangent lines. Therefore, two right angles are present in the quadrilateral. Making the angle relation: $\alpha = \frac{\theta}{2}$

Now, using right triangle trigonometry, it is possible to find delta X and delta Y by assuming that is the hypotenuse. This estimate is usually enough to achieve optimal precision as the computer can do these calculations hundreds of times per second. Therefore, in a time frame of a few milliseconds, the arc traveled is so small that it can be considered a straight line. However, over a longer period of time the error from this assumption could begin to accumulate and become significant. The tracking should be maximally precise and this is especially relevant if a slower computer is used. Fortunately, using mathematics, the exact value of the hypotenuse can be found



By knowing θ and r_C it is possible to find by using the cosine rule.

$$c^2 = a^2 + b^2 - 2ab\,cosC$$

In this case of an isosceles triangle, a and b are equal to r_C.

$$h = \sqrt{2r^2_C - 2r^2_C cos\theta}$$

However, it is important to check whether θ is equal to 0, which means the robot is driving straight, as otherwise h would then equal 0 as well. The algorithm checks if θ equals to 0 and if it does, then h set to d_C. When the robot is driving straight, d_C is a straight line and therefore, is the same as h.

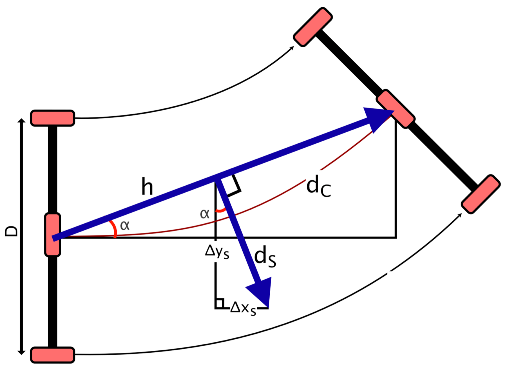
It is important to add the initial angle Φ_i, which is the angle the robot was oriented in, at the beginning of the time frame, in order to preserve the previous changes in orientation of the robot.

$$\Delta x = h \times cos(\Phi_i + \alpha)$$

$$\Delta y = h \times sin(\Phi_i + \alpha)$$

Perpendicular third wheel correction

As described before the robot has a third perpendicular tracking wheel in the middle. Because this wheel is exactly in the middle and is perpendicular, it does not rotate when the robot turns or drives straight. However, if the robot is pushed to the side or the robot is using mecanum wheels that allow it to drive sideways, then the two parallel wheels are unable to detect that. But, the third perpendicular wheel detects this shift.



As seen in figure 8, *dS* and *h* can be considered vectors. Because the wheel is placed perpendicularly to the two parallel wheels vectors *dS* and *h* are also perpendicular.

When a right angle triangle with *dS* as a hypotenuse is drawn the angle at the beginning of the vector *dS* is the same as the angle alpha, shown previously. Therefore, to account for the additional displacement, the shift of the robot adds to the overall displacement, basic right angle triangle trigonometry can be used almost in the same way as with the parallel wheels. The only difference is the switch of cos and sin because as seen in figure 8 the angle faces the *x* displacement axis.

$$\Delta x_s = d_S \times sin(\Phi_i + \alpha)$$

$$\Delta y_s = d_S \times cos(\Phi_i + \alpha)$$

So, when updating the global x, y and Φ coordinates of the robot in a cartesian system, both the parallel wheels and the perpendicular wheel's tracking information has to be added, together with the displacement from the last calculation:

$$\frac{a}{360} \times C = d_{R \vee L \vee S}$$

$$d_C = \frac{d_R + d_L}{2}$$

$$\theta = \frac{d_R - d_L}{D}$$

$$\alpha = \frac{\theta}{2}$$

$$h = \sqrt{2r_C^2 - 2r_C^2 cos\theta}$$

$$\Delta x = h \times cos(\phi_i + \alpha)$$

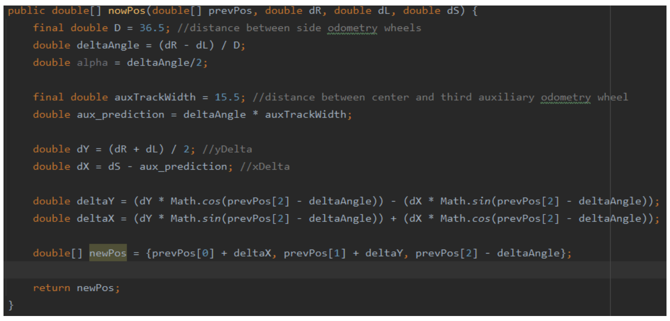
$$\Delta y = h \times sin(\phi_i + \alpha)$$

$$\Delta x_s = d_S \times sin(\phi_i + \alpha)$$

$$\Delta y_s = d_S \times cos(\phi_i + \alpha)$$

$$y = y + \Delta y + \Delta y_s \, x = x + \Delta x + \Delta x_s$$

This math is implemented in the following function which we use to update the robot's position based on dead-wheel encoder inputs:



```java
public double[] nowPos(double[] prevPos, double dR, double dL, double dS) {
    final double D = 36.5; //distance between side odometry wheels
    double deltaAngle = (dR - dL) / D;
    double alpha = deltaAngle/2;

    final double auxTrackWidth = 15.5; //distance between center and third auxiliary odometry wheel
    double aux_prediction = deltaAngle * auxTrackWidth;

    double dY = (dR + dL) / 2; //yDelta
    double dX = dS - aux_prediction; //xDelta

    double deltaY = (dY * Math.cos(prevPos[2] - deltaAngle)) - (dX * Math.sin(prevPos[2] - deltaAngle));
    double deltaX = (dY * Math.sin(prevPos[2] - deltaAngle)) + (dX * Math.cos(prevPos[2] - deltaAngle));

    double[] newPos = {prevPos[0] + deltaX, prevPos[1] + deltaY, prevPos[2] - deltaAngle};

    return newPos;
}
```